

# Die Linux Shell

Eine kurze Wiederholung / Einführung.

---

**Dozent:** Prof. Dr. Michael Eichberg  
**Kontakt:** michael.eichberg@dhbw.de  
**Version:** 1.0.1

---

**Folien:** [HTML] <https://delors.github.io/lab-shell/folien.de.rst.html>  
[PDF] <https://delors.github.io/lab-shell/folien.de.rst.html.pdf>  
**Fehler melden:** <https://github.com/Delors/delors.github.io/issues>

---

# 1. Erste Schritte

# Einloggen auf einem Server

## Ablauf

0. Starten Sie ggf. den CISCO Secure Client, wenn Sie auf interne Server der DHBW Mannheim zugreifen wollen. Dies ist ggf. auch aus dem WLAN der DHBW heraus notwendig!
1. Starten Sie eine Shell (z. B. Terminal oder iTerm auf MacOS bzw. Linux; cmd oder Powershell unter Windows.)
2. Verwenden Sie SSH, um sich auf dem Server einzuloggen. Sollte Ihr Accountname zum Beispiel `mueller` sein und der Server unter der IP `141.72.12.103` erreichbar sein, dann können Sie sich wie folgt auf dem Server einloggen, wenn passwort-basierte Authentifizierung möglich ist<sup>[1]</sup>:  

```
ssh -l mueller 141.72.12.103
```

---

SSH ist die Secure Shell und eine allg. etabliertes Werkzeug im Bereich der Administration von (Server-)Systemen, IoT Geräten etc.

Im professionellen Umfeld würde man zur Authentifizierung mittels SSH auf Zertifikate setzen. Für diese erste Übung verzichten wir darauf.

---

<sup>[1]</sup> Erhalten Sie eine Meldung, dass eine Passwort basierte Authentifizierung nicht erlaubt ist, dannn müssen Sie sich an Ihren Administrator wenden.

# Dateisysteme auf Unix-ähnlichen Betriebssystemen

- Die Verzeichnisstruktur in Unix-ähnlichen Systemen ist hierarchisch aufgebaut und beginnt bei der Wurzel (/).
- Alle Dateien und Verzeichnisse sind Unterknoten dieses Wurzelverzeichnisses.

## Hinweis

In Unixoiden Betriebssystemen trennt `/` auch Verzeichnisse.

## Beispiel

D. h. `/Users/administrator/.zsh_history` ist der Pfad der Datei `.zsh_history` im Verzeichnis `administrator`, welches wiederum im Verzeichnis `Users` zu finden ist. `Users` ist ein direktes Kindverzeichnis des Wurzelverzeichnisses `/`.

- Im Regelfall sind Datei- und Verzeichnisnamen *Case-sensitive*.

## Beispiel

Es ist möglich zwei Dateien `Test.txt` und `test.txt` gleichzeitig zu haben, die sich nur in der Groß-Kleinschreibung unterscheiden.

- Geräte (wie zum Beispiel Festplatten, USB Sticks, Zufallszahlengeneratoren ...) können an *fast beliebiger Stelle* eingehängt werden.

## Beispiel

Mittels `mount` kann man sich anzeigen lassen welche Geräte wo eingehängt sind:

```
1 /dev/vda2 on / type ext4 (rw,relatime)
2 /dev/vda1 on /boot/efi type vfat (rw,relatime,...)
```

- Systeminformationen sind auch direkt über spezielle Verzeichnisse verfügbar

## Beispiel

```
more /proc/cpuinfo
```

Gibt detaillierte Informationen über die CPU aus.

```
cat /proc/sys/kernel/printk
```

Gibt die Log-level Konfiguration des Kernels aus.

- Die Systemkonfiguration kann direkt über das Schreiben in spezialisierte Dateien erfolgen (meist sind Administratorrechte notwendig)

## Beispiel

```
echo 7 > /proc/sys/kernel/printk
```

Setzt den Logging-level für das Kernel Logging auf den Debug-Level, in dem der Wert 7 in die Datei `printk` geschrieben wird.

---

`cat`, `more` oder auch `less` sind ganz generische Programme zum Lesen von jeder Art von Text-basierten Dateien.



# Wichtige Verzeichnisse

Die wichtigsten Verzeichnisse - insbesondere unter Linux Betriebssystemen - sind:

Verzeichnis	Bedeutung / Inhalt
/	Wurzelverzeichnis – Ausgangspunkt aller Pfade
/bin	Wichtige Systemprogramme (z. B. <code>ls</code> , <code>cp</code> , <code>mv</code> )
/dev	Geräte-dateien (z. B. Festplatten: <code>/dev/sda</code> , Terminals: <code>/dev/tty</code> )
/etc	Systemweite Konfigurationsdateien
/home	Persönliche Benutzerverzeichnisse (z. B. <code>/home/alex</code> )
/lib	Wichtige Systembibliotheken
/media	Einhängepunkte für Wechseldatenträger
/mnt	Temporärer Einhängpunkt für Administratoren
/root	Heimatverzeichnis des Superusers <code>root</code>
/sbin	Systemprogramme für den Administrator
/tmp	Temporäre Dateien
/usr	Sekundäres Hierarchiesystem für Benutzerprogramme
/var	Veränderliche Dateien (z. B. Logs, Mails, Spool-Dateien)
~	Alias für das Homedirectory des aktuellen Nutzer.

# Grundlegendes

## Verzeichnis Operationen

- ls:** Gibt Informationen über eine Datei/Verzeichnis aus. Ermöglicht es insbesondere den Inhalt eines Verzeichnisses aufzulisten. `ls -al` gibt weitergehende Informationen aus. Insbesondere alle versteckten Dateien und Verzeichnisse sowie die Größe der Dateien und die Rechte.
- cd:** ( *change directory*)  
Wechsel des Verzeichnisses. Wechsel zu einem bestimmten Verzeichnis erfolgt durch Angabe des Names (`cd Test`). Wechsel zum Elternverzeichnis erfolgt über: `cd ..` (`..` steht immer für das Elternverzeichnis). Wechsel zum Wurzelverzeichnis erfolgt über: `cd /`
- mkdir:** Anlegen eines Verzeichnisses (`mkdir Test`)
- rmdir:** Löschen eines *leeren* Verzeichnisses (`rmdir Test`)
- rm:** ( *remove*) Löschen einer Datei (`rm Hello.txt`)

## Allgemeines

- Wenn man in Linux/MacOS ein Stern benutzt, dann wird dies zu den Namen aller nicht-versteckten Dateien/Verzeichnisse im aktuellen Verzeichnis aufgelöst und diese werden an das Programm übergeben! Zum Beispiel löscht `rm *` alle Dateien im aktuellen Verzeichnis.
- Verfügbarer Speicher auf der Festplatte: `df -h`
- Benutzer Speicher (eines Verzeichnisses): `du -h -s ~`
- Inhalt einer Datei editieren ist (zum Beispiel) mit `pico` möglich. Alternative kann mit Hilfe von `echo` ein Text in eine Datei umgeleitet werden (`echo "Hello" > Hello.txt`).
- Inhalt einer Datei ansehen: `cat`, `less`, `more`, ...

# Hilfe bekommen

In unixoiden Betriebssystemen gibt es für Kommandozeilentools Insbesondere die folgenden *eingebauten* Hilfemöglichkeiten:

- Hilfe des Programms (typischerweise `--help` oder einfach starten).
- **Manpages** (mittels `man <Name des Programms>`)
- Infopages (mittels `info <Name des Programms>`)

## Bemerkung

Manpages existieren für praktisch alle Kommandozeilenwerkzeuge.

Die schnelle Hilfe ist insbesondere bei „neueren“ Programmen nicht immer über `--help` verfügbar.

Je nach Anwendung kann das Starten der Anwendung ohne Parameter dazu führen, dass das Programm auf Eingaben wartet (z. B. `cat`).

# Kommandozeilenprogrammen beenden

<code>&lt;ctrl&gt;+c:</code>	Programm abbrechen (ggf. unkontrolliert)
<code>&lt;ctrl&gt;+d:</code>	Eingabe abschließen bzw. Ende der Eingabe
<code>kill &lt;PID&gt;:</code>	Mittels des Befehls <code>kill</code> lässt sich das Programm auch beenden. Erfordert ggf. dass man einen zweiten Terminal startet um die PID zu ermitteln (zum Beispiel mittels <code>top</code> oder <code>ps</code> ):



# Übung - Starten eines kleinen WebServers

## 1.1. Unix - Erste Schritte

1. Loggen Sie sich auf dem Server (z.B. 141.72.12.103) unter Verwendung der Zugangsdaten für Ihre Gruppe ein.
2. Erzeugen Sie ein Verzeichnis mit Ihrem Nachnamen (z.B. "eichberg")
3. Legen Sie in dem Verzeichnis eine rudimentäre `index.html` Datei an.

(Sie können zum Beispiel `pico`, `vim` oder `echo` verwenden!)

4. Starten Sie einen Webserver mit dem gerade angelegten Verzeichnis als Wurzel. Wählen Sie eine zufälligen Port `XXXX > 1024` aus.

(Zum Beispiel können Sie den bereits installierten Webserver mittels `http-server` wie folgt starten. Im Folgenden ist der Port `xxxx` und das Wurzelverzeichnis für Ihre Webseiten "eichberg":

```
http-server -p XXXX eichberg).
```

5. Wechseln Sie in Ihren Browser und öffnen sie die entsprechende Webseite.  
(Denken Sie daran, dass dieser Server nur `http` und nicht `https` unterstützt.)
6. Beenden Sie den WebServers
7. Löschen Sie das gerade angelegte Verzeichnis.

# Übung - Daten transferieren

## 12. Secure Copy (scp) nutzen

Nutzen Sie das Programm scp, um eine HTML Datei von Ihrem lokalen Rechner auf den Zielsever zu kopieren.


1. Nutzen Sie die Hilfe zu scp, um herauszufinden wie Sie eine Datei auf den Server in das richtige Verzeichnis kopieren können.
2. Öffnen Sie Ihren Browser, um zu prüfen, dass Sie auf die kopierte Webseite zugreifen können.

## 2. (Text-)Dateien und Ihre Bearbeitung

# Die grundlegenden Datenströme

Jedes Programm hat immer Zugriff auf die drei Standarddatenströme:

- `stdin` - Eingabedatenstrom.
- `stdout` - Ausgabedatenstrom für „normale“ Nachrichten.
- `stderr` - Ausgabedatenstrom für Fehlermeldungen.

Diese Datenströme sind (auch nur) Dateien, mit einem festgelegten Dateideskriptor ( *file descriptor*):

0:	<code>stdin</code>
1:	<code>stdout</code>
2:	<code>stderr</code>

# Umleitung von `stdin` in Dateien

> leitet die Ausgabe eines Programmes/Kommandos in eine Datei um; löscht bzw. legt die Zielfeile bei Bedarf an.

>> leitet die Ausgabe eines Programmes/Kommandos in eine Datei um; hängt die Ausgabe an das Ende einer bestehenden Datei an bzw. legt die Zielfeile bei Bedarf an.

## Beispiel

Erzeugen einer Datei `tmp/0s.txt`, die 1024 mal den Wert 0 in Base64 Kodierung enthält.

```
1 dd if=/dev/zero bs=1 count=1024 | base64 \ # 1024 * "0" in base64
2 > \ # Umleitung der Ausgabe
3 /tmp/0s.txt \ # in /tmp/0s.txt
```

## Bemerkung

Der `\` wird dazu verwendet, lange Shellkommandos über mehrere Zeilen schreiben zu können.  
Die `#` leitet ein "End-of-line Comment" ein.

# Umleiten von bestimmten Ausgaben an Dateien

Beim Umleiten von Ausgaben an eine Datei, kann der Dateideskriptor angegeben werden: [FD]><Ziel>  
2> leitet z. B. die Fehlerausgaben (🖨️ *stderr*) eines Programmes/Kommandos in eine Datei um;  
löscht bzw. legt die Zieldatei bei Bedarf an.

## Beispiel

Finden von bestimmten Dateien; aber Fehlerausgaben während des Suchprozesses ignorieren.

```
1 find / -iname "*txt*" -type f \  
2 2>/dev/null          # Umleitung aller Fehler nach /dev/null
```

# Grundlegende Prinzipien: Lesen aus einer Datei

< liest den Inhalt einer Datei und leitet diesen an das Programm/Kommando weiter; d. h. stellt den Inhalt über stdin zur Verfügung.

## Beispiel

Finden aller Städte, die mit "B" beginnen.[2]

```
1 grep B \           # filtert alle Zeilen, die ein "B" enthalten
2   < Big\ Cities.txt # der Inhalt von Big Cities.txt wird über stdin
3                   # zur Verfügung gestellt
```

[2] In diesem Fall könnte die Datei (`Big Cities.txt`) auch direkt als Parameter an `grep` übergeben werden. In anderen Fällen ist dies aber nicht möglich.

# Linux Shell - Grundlegendes Design-Pattern: Pipes and Filters

- Grundlegendes Konzept bzw. Entwurfsmuster (📁 *Design-Pattern*) in Unix-basierten Betriebssystemen.
- Ermöglicht die effiziente Verkettung von Befehlen. Die „Pipes-and-Filter“ Architektur erlaubt es komplexe Verarbeitungsoperationen mit Hilfe der Kombination von einfachen Befehlen durchzuführen.

| : Verbindet den Ausgabestrom (`stdout`) des vorhergehenden Befehls mit dem Eingabestrom (`stdin`) des nachfolgenden Befehls.

## Beispiel

Konvertierung des Wortes `Test` in Base64 Kodierung.

```
1 echo -n "Test" \  
2 |             \ # Weiterleitung der Ausgabe von echo an base64.  
3 base64
```

---

„Filter“: Kommandos/Programme, die von `stdin` lesen und nach `stdout` schreiben.



# Wichtige Linux Kommandozeilenwerkzeuge für die Verarbeitung von Passwortkandidaten

cat:	Dateien verketten.
sed:	Strom Editor.
grep:	Mustersuche auf Dateien.
tr:	Ersetzung und Löschung von Zeichen.
uniq:	Filtert wiederholte aufeinanderfolgende Zeilen in einer Datei.
sort:	Sortiert Dateien.
echo:	Schreibt Argumente auf <i>Standard Out</i> ( <code>stdout</code> ).
wc:	Zählt die Zeichen, Wörter, Zeilen einer Datei.
comm:	Vergleicht sortierte Listen und filtert entsprechend.
find:	Auswertung eines Ausdrucks für jede Datei während eines rekursiven Abstiegs über den Verzeichnisbaum.
awk:	Muster-orientierte Verarbeitung der Zeilen einer Eingabedatei.
base64:	(De-)Kodierung von Daten in Base64 Kodierung.
rev:	Dreht die Reihenfolge der Zeichen einer Zeile um.
head:	Zeigt die ersten ( <code>-n</code> ) Zeilen einer Datei an.
tail:	Zeigt die letzten ( <code>-n</code> ) Zeilen einer Datei an. ( <code>-f</code> folgt der Datei, d. h. wartet auf weitere Daten, die der Datei hinzugefügt werden.)

---

## Anwendungsfälle

Typischerweise werden diese Werkzeuge bei der Verarbeitung von Leaks/Aufbereitung von Wörterbüchern im Vorfeld gebraucht - vor dem eigentlichen Versuch das Passwort wiederherzustellen.

# echo

- Universell eingesetzt, um Inhalte in Dateien zu schreiben bzw. anzuhängen.
- `-n` um das automatische Anhängen von Zeilenumbrüchen zu unterdrücken. (Besonders dann wichtig, wenn man Hashes für Testdaten generieren will.)
- Entweder ein explizites Programm oder ein in die Shell eingebautes Kommando.


**Anwendungsfall:** Programmatisch Daten nach `stdout` schreiben.

```
1 $ echo -n "TestPasswort"
2 | shasum -a 256
3 | sed -E 's/ -$//'
4 2214db3d6fca761041242b9fc41fdcca
5 f0b2c7f556b80c0a91cfe6994437d807
```

## Bemerkung

Der hier zu sehende Befehl `shasum -a 256` ist unter einigen Linuxdistributionen einfach `sha256sum`.

# cat

- Liest alle Dateien sequentiell ein und schreibt diese auf `Standard Out (stdout)`.
- "-" repräsentiert `Standard In (stdin)`; dies ermöglicht die Verwendung von cat mitten in einer Verarbeitungskette.
- Liest (ggf.) von `stdin` bis zur EOF  *End-of-File* Markierung.

(Das Einlesen von der Kommandozeile kann mit `CTR+D` beendet werden.)

**Anwendungsfall:** Mehrere Teilwörterbücher sollen zusammengefügt werden.

Inhalt von `Test1.txt`: `Test1`

Inhalt von `Test2.txt`: `Test2`

```
1 $ echo "Test3" | cat Test1.txt Test2.txt -
2 Test1
3 Test2
4 Test3
```

# tr

■ Kopiert die Eingabe von `stdin` nach `stdout` und führt dabei Substitutionen und Löschungen durch.

**Anwendungsfall:** bestimmte Buchstaben - zum Beispiel Sonderzeichen - sollen gelöscht werden.

```
1 $ echo -n 'ab.cd_12!' | tr -dc '[:alnum:]' # -dc = delete complement
2 abcd12
```

**Anwendungsfall:** Groß- in Kleinbuchstaben verwandeln.

```
1 $ echo -n 'STARK' | tr '[:upper:]' '[:lower:]'
2 stark
```

# uniq

- vergleicht nebeneinanderliegende Zeilen und schreibt jede einzigartige Zeile einmal nach `stdout`. Nicht-nebeneinanderliegende Wiederholungen werden nicht erkannt.
- `-c` erlaubt es die Anzahl der Wiederholungen zu zählen.

**Anwendungsfall:** Wir möchten eine alphabetisch sortierte Liste nach der Häufigkeit des Vorkommens eines Wortes sortieren.

Mittels `uniq` kann die Häufigkeit gezählt werden.



Die Sortierung - zum Beispiel angefangen mit den am häufigsten vorkommenden Einträgen - kann danach im Nachgang erfolgen.

```
1 $ echo "Test\nTest\nSchlaraffenland\nTest" | uniq -c
2 2 Test
1 1 Schlaraffenland
1 1 Test
```

# awk

- Muster-orientierte Verarbeitung der Zeilen einer Eingabedatei.
- Jede Zeile wird segmentiert (Standardmäßig basierend auf Leerzeichen), die einzelnen Segmente werden mit `$1`, `$2`, ... bezeichnet. `$0` steht für die ganze Zeile.
- Die Verarbeitung erfolgt durch Muster-Handlungsanweisungen der Form:

```
pattern { action }
```

ist das Muster (  *pattern*) leer, dann wird die Zeile immer verarbeitet; ist keine Handlungsanweisung (  *action*) angegeben, dann wird die Zeile ausgegeben.

**Anwendungsfall:** Die Einträge einer Datei sollen nach länge sortiert werden. In diesem Fall, kann mit Hilfe von awk jede Zeile mit der Länge ausgegeben werden. Danach kann die Liste entsprechend sortiert werden.

```
1 $ echo "Test\nSchlaraffenland" | awk '{print length " " $1}'
2 4 Test
3 15 Schlaraffenland
```

# sort

- Sortiert eine Liste gemäß der entsprechenden Felder.
- `-r` sortiert in absteigender Reihenfolge.
- `-n` der Wert des ersten Feldes wird als numerischer Wert interpretiert.
- `-k` spezifiziert das Feld, nach dem sortiert werden soll. (z. B. `-k 3`)
- `-t` spezifiziert das Trennzeichen, das die Felder trennt. (z. B. `-t ','`)

**Anwendungsfall:** Sortiere eine Liste nach Häufigkeit des Vorkommens eines Wortes.

```
1 $ echo "abc\nxyz\nuvw\nxyz" \  
2 | sort \  
3 | uniq -c \  
4 | sort -nr \  
5 | sed -E 's/[0-9]+ *//' # entferne den Zähler  
6 xyz  
7 uvw  
8 abc
```

---

## Komplexes Beispiel

Sortierung einer Liste von Worten in absteigender Reihenfolge bzgl. (1) der Häufigkeit und (2) Länge.

```
1 $ printf '%s' "abc\nuvw\nxyz\nlmnop\nxyz\nuvw" \  
2 "nlmnop\nlmnop\nxyz\nncd\nncd\nncd" \  
3 | awk '{print length " " $1}'  
4 | sort  
5 | uniq -c  
6 | sort -nr -k 1 -k 2  
7 3 5 lmnop  
8 3 3 xyz  
9 3 2 cd  
10 2 3 uvw  
11 1 3 abc
```

Sortierung einer Liste von Worten in absteigender Reihenfolge bzgl. (1) der Häufigkeit und (2) aufsteigend bzgl. der Länge.

```
1 $ echo "abc\n" "uvw\n" "xyz\n" "lmnop\n" "xyz\n" "uvw\n" \  
2 "lmnop\n" "lmnop\n" "xyz\n" "cd\n" "cd\n" "cd" \  
3 | awk '{print length " " $1}' \  
4 | sort | uniq -c \  
5 | sort -k1nr -k2n  
6 3 3 cd  
7 3 4 xyz  
8 3 6 lmnop  
9 2 4 uvw  
10 1 3 abc
```

# base64

Base64 kodierte Werte bestehen nur noch aus gültigen ASCII Zeichen und können als "Text" gespeichert/übermittelt werden kann.

**Anwendungsfall:** In vielen Fällen können gehashte Passworte nicht roh (d. h. als Binärdaten) gespeichert werden sondern müssen **Base64** (oder vergleichbar) kodiert werden.

## Bemerkung

Je nach Betriebssystem/Konfiguration ist der Befehl unter Umständen etwas anders benannt. Grundsätzlich gibt es den Befehl auf allen Unixoiden.

```
1 # Codierung
2 $ echo "Dies_ist_ein_test" | base64
3 RGl1c19pc3RfZWluX3Rlc3QK
4 $ echo 'Dies_ist_ein_test!' | base64
5 RGl1c19pc3RfZWluX3Rlc3QhCg==
6
7 # Dekodierung
8 $ echo RGl1c19pc3RfZWluX3Rlc3QhCg== | base64 --decode
9 DHBW Mannheim
```



# grep

- Selektiert Zeilen, die einem gegebenen Muster entsprechen.
- -o gibt nur den Teil einer Zeile aus, der dem Muster entspricht.
- -v selektiert Zeilen für die kein Teil der Zeile dem Muster entspricht.
- -E erlaubt die Spezifikation von Mustern mit Hilfe von regulären Ausdrücken.
- -i ignoriert Groß-/Kleinschreibung (in Verbindung mit -E mgl. verwirrend).
- -P Perl kompatible Ausdrücke

**Anwendungsfall:** Alle Textfragmente in einem Leak finden, um danach mit Regeln neue Passwortkandidaten zu bilden.

```
1 $ echo "Test123\nmichael@dhbw.de\n345test@dhbw.de\nEnde__" \  
2 | grep -Eo "[a-zA-Z]{3,}" | sort -u  
3 Ende  
4 Test  
5 dhbw  
6 michael  
7 test
```

# sed - Stromeditor

- modifiziert die Eingabe gemäß der spezifizierten Kommandos in der angegebenen Reihenfolge.
- -E zur Verwendung moderner regulärer Ausdrücke
- Standardform: `Funktion[Agrumente]`
- Substitutionen: `s/Regulärer Ausdruck/Ersatz/[Kennzeichen]`; das Kennzeichen "g" z. B. bewirkt, dass jedes Vorkommen ersetzt wird; sonst nur das erste Vorkommen.

**Anwendungsfall:** Löschen des ersten Sonderzeichens in einer Zeile.

```
1 $ echo 'ab_cd!_ef?' | sed -E 's/[^a-zA-Z0-9]//'
2 abcd!_ef?
```

**Anwendungsfall:** Analyse der Struktur eines Leaks durch das Abbilden **aller** Buchstaben auf die Repräsentanten: l(lower) u(upper) d(digits) s(special).

```
1 $ echo 'aB_c1d!_ef?' |
2 sed -E -e 's/[a-z]/l/g' -e 's/[A-Z]/u/g' -e 's/[0-9]/d/g' -e 's/[^lud]/s/g'
3 lusldlsslsls
```

---

## Hinweis

sed auf dem Mac (BSD) und sed unter Linux (GNU) unterscheiden sich teilweise deutlich.

# find

- durchläuft den Dateibaum ab einer angegebenen Stelle und evaluiert dabei Ausdrücke.
- `-iname` Testet ob der Verzeichniseintrag - unabhängig von der Groß- und Kleinschreibung - dem gegebenen Muster entspricht.
- `-exec ... {} ... \;` ermöglicht es für jede gefilterte Datei `{}` einen Befehl auszuführen.

**Anwendungsfall:** Feststellen wie lange die Hashes sind.

```
1 $ find . -iname "*hash*" -exec wc -c {} \;
2 33 ./saltedmd5/hash.md5
3 38 ./saltedmd5/saltedhash.md5
4 129 ./scenario5/hash.sha125
5 65 ./scenario6/hash.sha256
6 65 ./scenario7/hash.sha256
7 65 ./scenario9/hash.sha256
```

# Software nachinstallieren

- Auf allen Linux und BSD Distributionen können Softwarepakete durch den Paketmanager des Betriebssystems nachinstalliert werden, z. B.:
  - `apt` (Debian, Ubuntu, Kali Linux, ...)
  - `yum` (RedHat, CentOS, ...)
  - `pacman` (Arch Linux, ...)
  - `brew` oder `macports` (MacOS) [**\***]

**Anwendungsfall:** Installieren von `ent` (ein Programm, das die Entropie von Dateien berechnet):

```
1 | sudo apt install ent
```

---

[**\***] Beide sind in diesem Fall nicht Teil des Betriebssystems, sondern müssen erst nachinstalliert werden, bevor damit weitere Software nachinstalliert werden kann.

# Shellprogrammierung

- Jede Shell (insbesondere: `zsh` (auf Mac und Kali Linux) und `bash` (Debian, Ubuntu, ...)) erlaubt es prozedurale Programme zu schreiben.

**Anwendungsfall:** Berechnung der Entropie für jede Datei in einer Liste.

```
1 #!/usr/bin/zsh                                # Shebang zur Spezifikation der Shell
2 IFS=$'\n'                                     # IFS = Internal Field Separator
3                                              # (Nur Zeilenumbrüche sind Trennzeichen)
4 rm Files.list.assessed                       # Lösche die Ausgabedatei
5 for i in $(cat Files.list); do               # Iteriere über die Zeilen in Files.list
6     echo "Processing: \"$i\"
7     ent -t "$i" | \                           # Berechne die Entropie
8     grep -E "^1" | \                         # Selektiere die Zeile mit der Entropie
9     tr -d '\n' | \                           # Lösche den Zeilenumbruch
10    cat - <(echo ", \"$i") \                 # Füge den Dateinamen hinzu
11    >> Files.list.assessed ; # Schreibe das Ergebnis
12 done;
```

# Fingerübungen

## Voraussetzung

Starten Sie z. B. Kali Linux (oder eine entsprechende VM), loggen Sie sich ein und starten Sie ein Terminal.

### 2.1. Dateien finden

Finden Sie die Datei, die die Standardpassworte von Postgres Datenbanken enthält (der Dateiname enthält sowohl `postgres` als auch `pass`).

---

## 2.2. MD5 Hash berechnen

Konkatenieren sie die Zeichenkette „MySalt“ (ohne Zeilenumbruch!) mit dem Inhalt von rockyou.txt (als Ganzes) und berechnen Sie davon den md5 Hash. Verwenden Sie keine expliziten Zwischenergebnisse.

---

## 2.3. Base64

Erzeugen Sie für eine Datei (z. B. `/usr/bin/wc`) einen MD5 hash und stellen Sie diesen der Datei selber voran bevor sie alles nach Base64 konvertieren.

---