

# Die Linux Shell - kurz Wiederholung

---

**Dozent:** Prof. Dr. Michael Eichberg  
**Kontakt:** michael.eichberg@dhbw-mannheim.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 auf Folien melden:**  
<https://github.com/Delors/delors.github.io/issues>

# 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*):

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

# Umleitung von `stdin` in Dateien

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

---

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

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

# 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.

```
find / -iname "*txt*" -type f \  
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.[#]\_

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

[1] 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.



# Wichtige Linux Kommandozeilenwerkzeuge für die Verarbeitung von Passwortkandidaten

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


```
$ echo -n "TestPasswort"  
  | shasum -a 256  
  | sed -E 's/ -$//'  
2214db3d6fca761041242b9fc41fdcca  
f0b2c7f556b80c0a91cfe6994437d807
```

## Hinweis

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**

```
$ echo "Test3" | cat Test1.txt Test2.txt -  
Test1  
Test2  
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.

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

---

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

```
$ echo -n 'STARK' | tr '[:upper:]' '[:lower:]'  
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.

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

```
$ echo "Test\nSchlaraffenland" | awk '{print length " " $1}'  
4 Test  
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.

```
$ echo "abc\nxyz\nuvw\nxyz" \  
| sort \  
| uniq -c \  
| sort -nr \  
| sed -E 's/ *[0-9]+ *//' # entferne den Zähler  
xyz  
uvw  
abc
```

13

## Komplexes Beispiel

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

```
$ printf '%s' "abc\nuvw\nxyz\nlmnop\nxyz\nuvw" \  
"\n\nlmnop\nlmnop\nxyz\nacd\nacd\nacd" \  
| awk '{print length " " $1}'  
| sort  
| uniq -c  
| sort -nr -k 1 -k 2  
3 5 lmnop  
3 3 xyz  
3 2 cd  
2 3 uvw  
1 3 abc
```

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

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

## Hinweis

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

```
# Codierung
$ echo "Dies_ist_ein_test" | base64
RGllc19pc3RfZWluX3Rlc3QK
$ echo 'Dies_ist_ein_test!' | base64
RGllc19pc3RfZWluX3Rlc3QhCg==

# Dekodierung
$ echo REhCVyBnYW5uaGVpb0== | base64 --decode
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.

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

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

---

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

```
$ echo 'aB_c1d!_ef?' |  
  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'  
lusldlsslsls
```

16

---

## 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.

```
$ find . -iname "*hash*" -exec wc -c {} \;  
33 ./saltedmd5/hash.md5  
38 ./saltedmd5/saltedhash.md5  
129 ./scenario5/hash.sha125  
65 ./scenario6/hash.sha256  
65 ./scenario7/hash.sha256  
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) <sup>[\*]</sup>
- 

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

```
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.

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

1. Starten Sie die Kali Linux VM, loggen Sie sich ein und starten Sie einen Terminal.
2. Finden Sie die Datei, die die Standardpassworte von Postgres Datenbanken enthält.

Tip der Dateiname enthält sowohl **postgres** als auch **pass**.

3. Konkatenieren sie die Zeichenkette "MySalt" (ohne Zeilenumbruch!) mit dem Inhalt von rockyou.txt und berechnen Sie davon den md5 Hash. Verwenden Sie keine expliziten Zwischenergebnisse.
4. 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.