HOWTO zur Standard-Ein/Ausgabe in der Shell

$Id: shell-stdio-HOWTO.txt,v 1.9 2025/02/23 20:14:55 tsbirn Exp $

Dieses Dokument beschreibt die Eigenschaften der Standard-Ein/Ausgabekanäle von Linux-Kommandos/Prozessen und ihre Steuerung per Umlenkung und Pipen durch die Shell.

INHALTSVERZEICHNIS

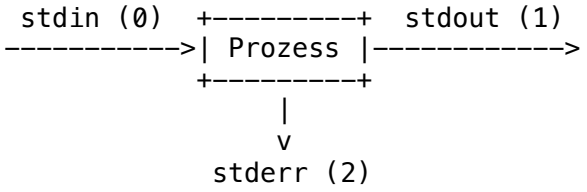
---

1) Standard-Kanäle

Jedes Linux-Kommando bzw. jeder aus einem Linux-Kommando erzeugte Linux-Prozess kennt 3 Standard-Kanäle zur Daten-Ein/Ausgabe, die von der Shell automatisch beim Erzeugen des Prozesses angelegt werden (zum Lesen bzw. Schreiben geöffnete Dateien):

Bezeichnung	Dateiname	N	Standard	Einsatzzweck
Standard-Eingabe	/dev/stdin	0	Tastatur	Eingaben
Standard-Ausgabe	/dev/stdout	1	Bildschirm	Normale Ausgaben (Daten)
Fehlerkanal	/dev/stderr	2	Bildschirm	Fehlermeldungen

Grafisch:



Diese 3 Kanäle sind standardmäßig wie angegeben mit der Tastatur oder dem Bildschirm verbunden, können aber per Umlenkung oder Pipe beliebig mit anderen Dateien oder Kommandos verbunden werden

Die Kommandos lesen dann nicht mehr von der Tastatur bzw. schreiben nicht mehr auf den Bildschirm, sondern lesen/schreiben von/auf Datei bzw. von/an andere Kommandos. sie bemerken die Umlenkung der Kanäle aber NICHT, da diese von der Shell eingerichtet werden, BEVOR das Kommando überhaupt gestartet wird.

HINWEIS: Die Shell ist ebenfalls nur ein Kommando, das Eingaben von stdin liest

(Kommandos) und Ausgaben auf stdout/stderr schreibt (beim Login eingerichtet). Die Ein/Ausgabekanäle eines Prozesses werden an seinen Kindprozess vererbt (in Shell aufgerufenes Kommando), falls beim Aufruf keine Umlenkung und kein Pipen erfolgt.

2) Filter und Pipeline

Programme, die Daten von der Standard-Eingabe lesen, sie filtern und manipulieren und auf der Standard-Ausgabe wieder ausgeben, werden auch "FILTER" genannt, weil sie wie ein "Kaffeefilter" arbeiten. Typische Filter sind z.B. folgende Programme:

```
grep, sort, uniq, tail, head, cut, paste, split, wc, sed, awk, tee, tr, ...
```

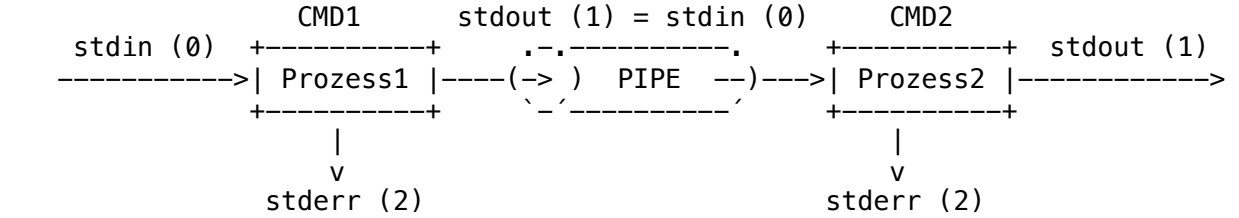
Mit Hilfe der "Filter" und Pipes kann das "BAUSTEINPRINZIP" von Linux sehr einfach realisiert werden: Mehrere über Pipes verbundene Kommandos ergeben ein kombiniertes (komplexes) Filter, eine sogenannten "PIPELINE" (Filterkette, Verarbeitungskette). Die Daten fließen vom 1. bis zum letzten Kommando durch die gesamte Verarbeitungskette und werden dabei schrittweise verarbeitet.

```
CMD1 | CMD2 | ... | CMDn-1 | CMDn
```

Eventuell wird die Eingabe des 1. Kommandos von Datei gelesen und die Ausgabe des letztes Kommandos auf Datei geschrieben:

```
CMD1 < INPUT | CMD2 | ... | CMDn-1 | CMDn > OUTPUT
```

Der Fehlerkanal jedes Kommandos wird dabei standardmäßig nicht umgelenkt. Die Fehlermeldungen erscheinen also auf dem Terminal, sie werden nicht an das nächste Kommando weitergeleitet:



HINWEIS: Pipes haben eine relativ kleine Größe (4/8/16 KByte), liegen im Speicher und SYNCHRONISIEREN über den Datenfluß damit verbundene Prozesse (laufen parallel). Sie belegen keinen Plattenplatz und sind extrem schnell.

HINWEIS: Da Linux keine Binärdaten kennt, sondern fast alle Daten in zeilenorientierten (ASCII)-Textdateien ablegt (Zeilen werden durch Newline "\n" begrenzt), werden ZEILENORIENTIERTE TEXTE als Standard-Datenformat von fast allen Linux-Kommandos gelesen und erzeugt.

3) Umlenkung (Redirection) und Pipen

Folgende Syntax zur Ein/Ausgabe-Umlenkung eines Kommandos CMD gibt es:

Syntax	Bedeutung
CMD < FILE	stdin von Datei FILE lesen
CMD << EOF	stdin ist MEHRZEILIGER TEXT bis EOF (end-of-file)
...TEXT...	beliebiger Text (Here-Dokument)
EOF	Text "EOF" von oben am Zeilenanfang!
CMD <<< TEXT	stdin ist String TEXT (Here-String)
CMD > FILE	stdout auf Datei FILE schreiben (vorher löschen!)
CMD >> FILE	stdout an Datei FILE anhängen
CMD 2> FILE	stderr auf Datei FILE schreiben (vorher löschen!)

CMD 2>> FILE	stderr an Datei FILE anhängen
CMD 1>&2	stdin zu stderr hinzufügen (kombinieren)
CMD 2>&1	stderr zu stdin hinzufügen (kombinieren)
CMD &> FILE	stdout+stderr auf Datei FILE schreiben (vorher leeren!)
	entspricht "> FILE 2>&1" oder "2> FILE 1>&2"
CMD >& FILE	stdout+stderr auf Datei FILE schreiben (analog)
CMD >>& FILE	stdout+stderr an Datei FILE anhängen (analog)
CMD1 | CMD2	stdout von CMD1 mit stdin von CMD2 verbinden
CMD1 2>&1 | CMD2	stdout+stderr von CMD1 mit stdin von CMD2 verbinden
CMD1 |& CMD2	stdout+stderr von CMD1 mit stdin von CMD2 verbinden
CMD >| FILE	stdout auf Datei FILE schreiben (auch bei "noclobber"!)

HINWEIS:

* Eine Umlenkung verknüpft ein Kommando mit einer Datei.
  Eine Pipe verknüpft zwei Kommandos miteinander.

* Nach dem Pipe-Symbol darf ein Zeilenumbruch erfolgen.
  Nach den Umlenk-Symbolen darf KEIN Zeilenumbruch erfolgen.

* Eine Umlenkung darf irgendwo vor/im/hinter dem Kommando stehen
  (üblich ist am Ende des Kommandos):

```
CMD OPT PARARM < FILE   # OK (typische Schreibweise)
< FILE CMD OPT PARARM   # OK (unüblich)
CMD OPT < FILE PARARM   # OK (unüblich)
```

* Eine Pipe muss zwischen 2 Kommandos stehen (diese werden von ihr verknüpft):

```
CMD1 OPT1 PARAM1 | CMD2 OPT2 PARAM2   # OK (einzeilig)
CMD1 OPT2 PARAM1 |                    # OK (zweizeilig)
  CMD2 OPT1 PARAM2                    #
```

* Ohne Eingabe-Umlenkung (stdin) lesen nahezu alle Kommandos (Filter) die nach
  dem Kommando angegebenen (Eingabe)Dateien automatisch ein:

```
CMD FILE...
```

* Ohne Angabe von Eingabe-Umlenkung, Pipe und Eingabe-Datei liest ein Kommando
  automatisch von der Tastatur. Das Dateiende bei der manuellen Eingabe ist
  durch "Strg/Ctrl-D" anzugeben.

* Der Name "-" als Dateiname steht für die Standard-Eingabe oder -Ausgabe:

```
CMD               # Liest von Standard-Eingabe (Tastatur)
CMD -             # Liest von Standard-Eingabe (Tastatur)
CMD /dev/stdin    # Liest von Standard-Eingabe (Tastatur)
CMD < /dev/stdin  # Liest von Standard-Eingabe (Tastatur)
```

* Die 3 Standard-Ein/Ausgabekanäle haben pro Prozess auch die festen Dateinamen
  "/dev/stdin", "/dev/stdout" und "/dev/stderr".

```
CMD                                          # Standardverhalten
CMD < /dev/stdin > /dev/stdout 2> /dev/stderr   # Analog
```

* Der Name "/dev/null" kann als Ausgabe-Datei angegeben werden, falls Ausgaben
  ignoriert (weggeworfen) werden sollen (verschluckt die Daten einfach):

```
CMD > /dev/null               # Standard-Ausgabe wegwerfen
CMD 2> /dev/null              # Standard-Fehlerausgabe wegwerfen
CMD > /dev/null 2> /dev/null  # Beide Standard-Ausgaben wegwerfen
```

Wird häufig zum IGNORIEREN der Fehlermeldungen angewendet oder zum WEGWERFEN der Ausgabe, falls nur der Exit-Status eines Kommandos relevant ist.

```
rm /tmp/FILE 2> /dev/null          # Fehlermeldungen wegwerfen
if CMD > /dev/null 2> /dev/null    # Nur Exit-Status relevant
then
    ...
fi
```

* Der Name "/dev/null" kann als (leere) Eingabe-Datei angegeben werden, falls diese für ein CMD notwendig ist, aber das Lesen sofort beendet werden soll:

```
CMD < /dev/null
```

3.1) Fehlermöglichkeiten

* Jedem Kanal ist per Umlenkung/Pipe nur EINE Quelle/Ziel zuordenbar:

```
grep "text" < datei1 datei2       # Fehler (zweideutig)
grep "text" < datei1              # OK     (eindeutig)
grep "text" datei2                # OK     (eindeutig)
cat /etc/passwd > datei | less    # Fehler (zweideutig)
cat /etc/passwd > datei           # OK     (eindeutig)
cat /etc/passwd | less            # OK     (eindeutig)
```

* Ein Filter kann eine Datei entweder nur lesen ODER schreiben, aber nicht beides gleichzeitig:

```
grep < datei > datei     # Falsch ("datei" anschliessend leer)
grep < datei > datei2    # OK
```

* Eine Datei kann nur für EINE Umlenkung als Ziel verwendet werden:

```
grep > datei 2> datei    # Falsch (Kanal 1 oder 2 gewinnt)
grep > datei 2> datei2   # OK (zwei verschiedene Dateien)
grep > datei 2>&1        # OK (eine Datei, Ausgabekanäle zusammenfügen)
grep 2> datei 1>&2       # OK (eine Datei, Ausgabekanäle zusammenfügen)
```

3.2) Beispiele für Dateiumlenkung

Kommando	Beschreibung
cat	stdin auf stdout schreiben (Tastatur -> Bildschirm)
cat > FILE	stin auf FILE schreiben (vorher leeren!)
cat < FILE	stdin von FILE lesen
cat < FILE1 > FILE2	FILE1 nach FILE2 kopieren
cat > FILE2 < FILE1	Analog (d.h. Reihenfolge der Umlenkungen egal!)
cat < FILE > FILE	FEHLER (Datei FILE ist anschließend leer!)
cat >> FILE	stdout an FILE anhängen
cat < FILE >> FILE	FEHLER (Datei FILE wird beliebig lang!)
cat 2> FILE	stderr auf FILE schreiben
cat < xyz 2> error	Fehlermeldung "Datei xyz unbekannt" am Bildschirm
cat 2> error < xyz	Fehlermeldung "Datei xyz unbekannt" in "error"
less FILE	Datei FILE seitenweise anzeigen
cat < FILE | less	Analog (2 Prozesse)
cat FILE | less	Analog (2 Prozesse)

HINWEIS: Nicht den "Useless use of cat award" versuchen zu gewinnen...
(CMD1 | cat | cat | cat | ... | cat | CMD2)

3.3) Beispiele für Pipes

Kommando	Beschreibung
ls -l /bin | sort	Datei nach Typ + Rechten sortieren
ls -l /bin | sort | less	... + seitenweise anzeigen
ls -l /bin | sort | head	... + die ersten 10 aufsteigend
ls -l /bin | head | sort	... + 10 beliebige aufsteigend
ls -l /bin | sort | tail	... + die letzten 10 aufsteigend
ls -l /bin | sort -r | head	... + die letzten 10 absteigend
ls -l /bin | sort -r | head | sort	... + die letzten 10 aufsteigend
ls -l /bin | sort | tail	... + die letzten 10 aufsteigend
ls -l /bin | sort | tail | sort -r	... + die letzten 10 absteigend
ls -l /bin | sort +4nr	Dateien nach Größe sortieren
ls -l /bin | sort +4nr | head	... + die größten 10 absteigend
ls -l /bin | sort +4n | head	... + die kleinsten 10 aufsteigend

3.4) Option "noclobber"

Per Option "noclobber" (übersetzt "nicht zusammenschlagen") kann verhindert werden, dass bereits existierende Dateien versehentlich per Dateiumlenkung überschrieben werden:

```
set -o noclobber
```

Das Zurücksetzen der Option erfolgt mittels (+ = minus "durchgestrichen"):

```
set +o noclobber
```

Beispiel:

```
echo "aaa" > FILE      # OK
echo "bbb" > FILE      # OK (FILE wird überschrieben)
echo "ccc" >> FILE     # OK (FILE wird verlängert)
set -o noclobber       # Option "noclobber" setzen
echo "ddd" > FILE      # Klappt NICHT (FILE wird nicht überschrieben)
echo "eee" >> FILE     # OK (FILE wird verlängert)
echo "ddd" >| FILE     # OK (FILE wird überschrieben - erzwungen!)
set +o noclobber       # Option "noclobber" löschen
echo "fff" > FILE      # OK (FILE wird überschrieben)
echo "ggg" >> FILE     # OK (FILE wird verlängert)
```

4) Here-Dokument

Um Kommandoaufrufe und Daten in einem Skript gemeinsam pflegen zu können, ist die Angabe von Daten zu einem Kommando direkt im Skript beim Kommandoaufruf als sogenanntes "Here-Dokument" möglich. Das folgende Beispiel (EOF = "end of file")

```
sort << EOF
...Text...
...Text...
...Text...
EOF
```

übergibt dem sort-Kommando die Textzeilen zwischen den beiden "Begrenzern" "EOF" zum Sortieren. Das Begrenzer-Wort "EOF" ist frei wählbar und muss als Dokumentabschluss auf einer Zeile für sich alleine am Zeilenanfang stehen, damit es erkannt wird.

Im Text werden Variablen- ($VAR) und Kommando-Substitutionen (`CMD...` bzw. $(CMD...) durchgeführt. Gibt man VOR dem Begrenzer einen Backslash an oder

setzt ihn in "..." oder '...', dann finden diese Ersetzungen nicht statt:

```
sort << \EOF          sort << "EOF"          sort << 'EOF'
...Text...            ...Text...             ...Text...
...Text...            ...Text...             ...Text...
...Text...            ...Text...             ...Text...
EOF                   EOF                    EOF
```

Ein "-" nach dem "<<" erlaubt das Einrücken der Datenzeilen mit Tabulatoren, diese Tabulatoren werden bei der Übergabe an das Kommando ignoriert:

```
sort <<- EOF
<TAB> Text...
<TAB> Text...
<TAB> Text...
EOF
```

HINWEIS: Probiert man diese Umlenkung interaktiv auf der Kommandozeile aus, so gibt die Shell nach der 1. Zeile bis zum EOF den sogenannten "Fortsetzungsprompt" ">" aus (definiert in Variable "PS2").

Die Weiterverarbeitung der Ergebnisdaten, die von einem Kommando (hier "sort) aus Here-Dokument-Daten erstellt werden an ein weiteres Kommando (hier "uniq") erfolgt folgendermaßen:

```
sort << EOF | uniq | ...
...Text...
...Text...
...Text...
EOF
```

4b) Here-String

Ein Here-String wird über die Standard-Eingabe an ein Kommando weitergegeben:

```
CMD <<< "Here String"          # Kein neuer Prozess
echo "Here String" | CMD       # Analog, aber neuer Prozess
```

5) Linux-Filterprogramme

Filterprogramme lesen ZEILENORIENTIERTE TEXTDATEN von der Standard-Eingabe, verarbeiten sie gemäß den angegebenen Optionen und/oder Argumenten und geben die bearbeiteten Daten auf der Standard-Ausgabe wieder aus. Durch Pipe-Symbole können beliebig viele dieser Filter miteinander verbunden werden, um aus einfachen Programmen eine leistungsfähige Verarbeitungskette ("Pipeline") zu kombinierten, die ein bestimmtes Problem schrittweise löst.

VORGEHEN: Üblicherweise werden derartige Verarbeitungsketten schrittweise Stück für Stück interaktiv in der Kommandozeile aufgebaut, indem immer wieder neue Filter an die bisherige Verarbeitungskette (Pipeline) angefügt werden. Ist die gewünschte Funktionalität erfolgreich realisiert, kann die Filterkette in einer Datei abgespeichert werden, um sie in Zukunft als Shell-Skript aufrufen zu können.

Standard-Filterprogramme (per Optionen änderbare feste Funktionalität):

Prog	Beschreibung
cat	Hängt Dateien aneinander, kopiert Eingabe [concatenate]
comm	Vergleicht 2 Dateien binär [compare]
cmp	Vergleicht 2 Dateien binär [compare]
column	Datenzeilen nebeneinander in mehreren Spalten anordnen
colrm	Spalten entfernen [column remove]

cut	Schneidet Textspalten aus
diff	Vergleicht 2 Dateien und zeigt Unterschiede an [difference]
fmt	Formatiert Texte auf bestimmte Zeilenlänge um [format]
head	Zeigt erste N Zeilen an [Kopf]
hexdump	Gibt in Binärformat aus [hexadecimal dump]
join	Verknüpft 2 Dateien über ein Schlüsselfeld
less	Blättert seitenweise durch Daten (modern)
more	Blättert seitenweise durch Daten (veraltet)
most	Blättert seitenweise durch Daten (modern)
nl	Numeriert Zeilen durch [number line]
od	Gibt Zeichencodes aus [octal dump]
paste	Zeilen mehrerer Dateien horizontal zusammenfassen (spalten)
shuf	Zeilen in zufälliger Reihenfolge ausgeben [shuffle]
sort	Sortiert Zeilen alphabetisch oder numerisch
split	Zerlegt Dateien in Blöcke
tail	Zeigt letzte N Zeilen an [Schwanz]
tac	Dreht Zeilen-Reihenfolge herum (1. <-> letzte, "cat" invers)
tee	Dupliziert Datei [T-Stück]
tr	Übersetzt Zeichen in andere Zeichen [translate]
uniq	Läßt doppelt vorkommende Zeilen weg [unique]
wc	Zählt Zeichen, Worte und Zeilen [word count]

Programmierbare Filterprogramme (mit Regulären Ausdrücken):

grep	Filtert Zeilen per Regex [global regular expression print]
egrep	Extended grep (erweiterte reguläre Ausdrücke)
fgrep	Fix grep (keine regulären Ausdrücke)
sed	Editiert Text mit angegebenen Kommandos [stream editor]
ed/ex	Editiert Textdatei mit angegebenen Kommandos [editor]

Typische Linux-Skript-Sprachen zur Realisierung komplexer Filterprogramme:

awk	Programmiersprache u.a. zur Textverarbeitung
perl	Programmiersprache u.a. zur Textverarbeitung
php	Programmiersprache u.a. zur Textverarbeitung
python	Programmiersprache u.a. zur Textverarbeitung
ruby	Programmiersprache u.a. zur Textverarbeitung
tcl/tk	Programmiersprache u.a. zur Textverarbeitung
lua	Programmiersprache u.a. zur Textverarbeitung

6) Große Beispiele

6.1) Worthäufigkeit ermitteln

Die folgende "Pipeline" besteht aus 8 Kommandos und selektiert die 10 seltensten Worte aus einer oder mehreren Textdatei(en):

```
cat FILE... |              # Angegebenen Dateien aneinanderhängen
    tr -c "A-Za-z" "\n" |  # Nichtbst. in Newline "\n" umwandeln [complement]
    tr "A-Z" "a-z" |       # GROSS- in Kleinbuchstaben umwandeln
    sed '/^ *$/d' |        # Leerzeilen löschen
    sort |                 # Worte sortieren
    uniq -c |              # Worthäufigkeiten ermitteln [count]
    sort -n |              # Nach Häufigkeit sortieren [numeric]
    head                   # Die ersten 10 Worte anzeigen
                           #   (auch sort -nr | tail)
```

Beschreibung: "cat" [concatenate] hängt die Dateien FILE... aneinander, "tr" [translate] tauscht Buchstaben gegen andere Buchstaben aus ("\n" ist das

Zeichen "newline"), "sed" [stream editor] editiert den Text mit den angegebenen Kommandos, "uniq" [unique] fasst gleiche Zeilen zusammen und zählt ihre Häufigkeit, "sort" sortiert alphabetisch oder numerisch und "head" zeigt die ersten 10 Zeilen an.

Durch Angabe folgender Optionen beim zweiten "sort" und bei "head" können die 10 häufigsten Worte oder eine andere Zahl als 10 Worte selektiert werden:

```
  -r   bei "sort": Sortiert absteigend --> häufigsten 10 Worte
  -nNN bei "head": Zeigt NN Zeilen an (Standard: 10)
  -nNN bei "tail": Zeigt NN Zeilen an (Standard: 10)
```

6.2) Nicht in Wörterbuch enthaltene Worte ausgeben

Die folgende "Pipeline" besteht aus 10 Kommandos und selektiert aus einer oder mehreren Textdatei(en) die 10 Worte, die NICHT im (kleingeschriebenen und alphabetisch sortierten) Wörterbuch DICT stehen (pro Zeile ein Wort).

```
  cat FILE... |               # Angegebenen Dateien aneinanderhängen
      tr -c "A-Za-z" "\n" |   # Nichtbst. in Newline "\n" umwandeln [complement]
      tr "A-Z" "a-z" |        # GROSS- in Kleinbuchstaben umwandeln
      sed '/^ *$/d' |         # Leerzeilen löschen
      sort |                  # Worte sortieren
      uniq |                  # Gleiche Worte zusammenfassen
      diff - DICT |           # Mit Wörterbuch DICT vergleichen ("-" = stdin)
      grep "^<" |             # Worte filtern, die nicht im Wörterbuch sind
      sed "s/^< *//" |        # Kennzeichen "<..." entfernen
      less                    # Seitenweise blättern
```

Beschreibung: Siehe oben + "diff" vergleicht 2 Dateien und zeigt die Unterschiede an, "grep" wählt zu einem Muster passende Zeilen aus, "less" blättert seitenweise durch die Daten.