HOWTO zur Standard-Ein/Ausgabe in der Shell

(C) 2004-2017 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
OSTC Open Source Training and Consulting GmbH
http://www.ostc.de

$Id: shell-stdio-HOWTO.txt,v 1.6 2017/06/28 18:11:34 tsbirn Exp $

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

INHALTSVERZEICHNIS

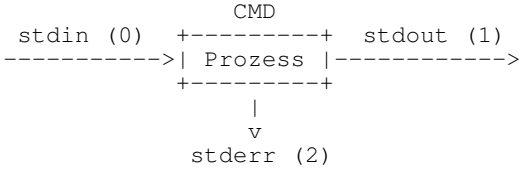
---

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). Ein/Ausgabekanäle werden an den Kindprozess vererbt (in Shell aufgerufenes Kommando), wenn 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 "Baukastenprinzip" 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 verarbeitet.

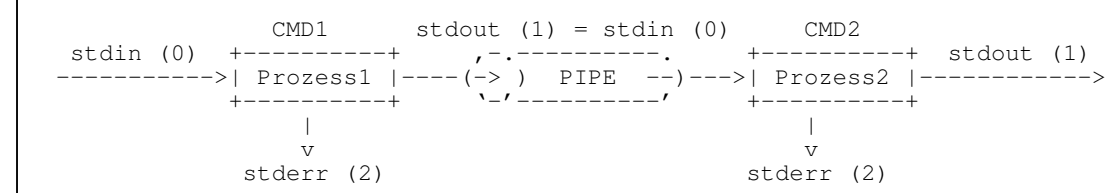
```
  CMD1 | CMD2 | ... | CMDn-1 | CMDn
```

Eventuell wird die Eingabe von Datei gelesen und die Ausgabe auf Datei geschrieben:

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

Der Fehlerkanal jedes Kommandos wird dabei 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 kaum Binärdaten kennt, sondern fast alle Daten in zeilenorientierten (ASCII)-Textdateien ablegt (Zeilen werden durch Newline "\n" begrenzt), werden solche Textdateien als Standard-Datenformat von fast allen Kommandos gelesen und erzeugt.

3) Umlenkung (Redirection) und Pipen

Folgende Syntax zur Ein/Ausgabe-Umlenkung gibt es:

Syntax	Bedeutung
CMD < FILE	stdin von Datei FILE lesen
CMD << EOF	stdin von CMD-Zeile bis Zeile mit EOF
...TEXT...	beliebiger Text (Here-Dokument)
...TEXT...	beliebiger Text (Here-Dokument)
EOF	Text "EOF" von oben am Zeilenanfang!
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" (bash)
CMD >& FILE	stdout+stderr auf Datei FILE schreiben (csh)
CMD1 | CMD2	stdout von CMD1 mit stdin von CMD2 verbinden
CMD1 2>&1 | CMD2	stdout+stderr von CMD1 mit stdin von CMD2 verb. (bash)
CMD1 |& CMD2	stdout+stderr von CMD1 mit stdin von CMD2 verb. (csh)
CMD >| FILE	stdout auf Datei FILE schreiben (auch bei noclobber!)

HINWEISE:

* Eine Umlenkung verknüpft ein Kommando und eine Datei,
  Eine Pipe verknüpft zwei Kommandos.

* Nach dem Pipe-Symbol darf ein Zeilenumbruch erfolgen,
  nach dem Umlenk-Symbol darf KEIN Zeilenumbruch erfolgen.

* Eine Umlenkung darf irgendwo vor/im/hinter dem Kommando stehen:

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

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

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

```
* Ohne Eingabeumlenkung (stdin) lesen praktisch alle Kommandos (Filter) die
  nach dem Kommando aufgelisteten (Eingabe)Dateien automatisch ein:

    CMD FILE...

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

* Der explizite Name "-" steht als Dateiname für Standardeingabe/ausgabe:

    CMD               # Liest von Standardeingabe (Tastatur)
    CMD -             # Liest von Standardeingabe (Tastatur)
    CMD /dev/stdin    # Liest von Standardeingabe (Tastatur)
    CMD < /dev/stdin  # Liest von Standardeingabe (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 (leere) Eingabedatei angegeben werden, wenn
  diese für ein CMD notwendig ist, aber das Lesen sofort beendet werden soll:

    CMD /dev/null

* Der Name "/dev/null" kann als Ausgabedatei angegeben werden, wenn Ausgaben
  ignoriert (weggeworfen) werden sollen (verschluckt die Daten einfach):

    CMD > /dev/null               # Standardausgabe wegwerfen
    CMD 2> /dev/null              # Standardfehlerausgabe wegwerfen
    CMD > /dev/null 2> /dev/null  # Beide Standardausgaben wegwerfen

  Er wird häufig zum Ignorieren der Fehlermeldungen angewendet oder wenn nur
  der Exit-Status eines Kommandos relevant ist (nicht aber seine Ausgaben):

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


3.1) Fehlermöglichkeiten
------------------------
* Jedem Kanal ist per Umlenkung/Pipe nur EINE Quelle/Ziel zuordenbar:

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

* Ein Filter kann eine Datei nur lesen ODER schreiben
  (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                 | stdout auf stdin schreiben (Tastatur -> Bildschirm) |
  | cat > FILE          | stdout auf FILE schreiben (vorher leeren!)          |
  | cat < FILE          | stdin von FILE lesen                                |
  | cat < FILE1 > FILE2 | stdin von FILE1 lesen, stdout auf FILE2 schreiben   |
  | cat > FILE2 < FILE1 | Analog (d.h. Reihenfolge der Umlenkungen ist 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...

3.3) Beispiele für Pipes

Kommando	Beschreibung
ls -l /bin | sort	Datei nach Typ und 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 +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"

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

```
set -o noclobber
```

Das Zurücksetzen der Option erfolgt mittels:

```
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)
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 er erkannt wird.

Im Datenteil werden noch Variablen- ($VAR) und Kommando-Substitutionen (`CMD...` bzw. $(CMD...) durchgeführt. Gibt man VOR dem Schlüsselwort einen Backslash an oder setzt das Schlüsselwort in "..." oder '...', 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 auf der Kommandozeile aus, so gibt die
Shell nach der 1. Zeile bis zum EOF den sogenannten "Fortsetzungsprompt" ">"
aus (definiert in Variable "PS2").

5) Linux-Filterprogramme

Filterprogramme lesen zeilenorientierte ASCII-Daten von der Standard-Eingabe,
verarbeiten sie gemäß den angegebenen Optionen und/oder Argumenten und geben die
bearbeiteten Daten auf der Standardausgabe wieder aus. Durch Pipe-Symbole
können beliebig viele dieser Filter miteinander verbunden werden, um aus
einfachen Programmen eine leistungsfähige Verarbeitungskette 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.

Bekannte Linux-Filterprogramme (relativ feste Funktionalität):

Prog	Beschreibung
cat	Hängt Dateien aneinander, kopiert Eingabe [concatenate]
comm	Vergleicht 2 Dateien binär
cut	Schneidet Textspalten aus
diff	Vergleicht 2 Dateien und zeigt Unterschiede an [difference]
fmt	Formatiert Texte auf bestimmte Zeilenlänge um [format]
head	Zeigt die ersten N Zeilen an [Kopf]
join	Verknüpft 2 Dateien über ein Schlüsselfeld
less	Blättert seitenweise durch Daten
more	Blättert seitenweise durch Daten
nl	Numeriert Zeilen durch [number line]
od	Gibt Zeichencodes aus [octal dump]
paste	Fasst Dateien horizontal zusammen (spaltenweise)
sort	Sortiert alphabetisch oder numerisch
split	Zerlegt Dateien in Blöcke
tail	Zeigt die letzten N Zeilen an [Schwanz]
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 expr print]
sed	Editiert Text mit angegebenen Kommandos [stream editor]
ed	Editiert Textdatei mit angegebenen Kommandos [editor]
awk	Programmiersprache zur Textverarbeitung
perl	Umfangreiche Programmiersprache u.a. auch zur Textverarbeitung
python	Umfangreiche Programmiersprache u.a. auch zur Textverarbeitung
ruby	Umfangreiche Programmiersprache u.a. auch zur Textverarbeitung

6) Grosse Beispiele

6.1) Worthäufigkeit ermitteln

Die folgende "Pipeline" besteht aus 8 Kommandos und selektiert aus einer ASCII-
Textdatei die 10 seltensten Wörter.

```
  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 Dateien die Dateien FILE... aneinander,
"tr" [translate] übersetzt Buchstaben ("\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 2. "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
  -NN bei "head": Zeigt NN Zeilen an (Standard: 10)
  -NN 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
ASCII-Textdatei 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 aus Wörterbuch weglassen
      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.
```