HOWTO zum Aufruf von externen Programmen in einem Perl-Skript.

(C) 2014 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
OSTC Open Source Training and Consulting GmbH
http://www.ostc.de

$Id: perl-ipc-HOWTO.txt,v 1.8 2019/11/26 19:37:07 tsbirn Exp $

Dieses Dokument beschreibt die verschiedenen Verfahren zum Aufruf von
Kommandos in einem Perl-Skript.

INHALTSVERZEICHNIS

---

1) Aufruf-Arten

FÃ¼r den Aufruf eines externen Kommandos CMD in einem Perl-Skript gibt es
folgende MÃ¶glichkeiten, je nach Anwendungszweck ist eine geeignete auszuwÃ¤hlen:

A u f r u f	STDIN	STDOUT	STDERR	Skript	Exit
system("CMD")	Geerbt	Geerbt	Geerbt	Wartet	RW
system("CMD &")	Geerbt	Geerbt	Geerbt	Wartet nicht	--
exec("CMD")	Geerbt	Geerbt	Geerbt	Wird ersetzt	(RW)
qx/CMD/ `CMD`	Geerbt	STRING	Geerbt	Wartet	$?
open(HANDLE, "CMD|")	Geerbt	HANDLE	Geerbt	Laeuft weiter	$?
open(HANDLE, "-|", "CMD")	Geerbt	HANDLE	Geerbt	bis close()	$?
open(HANDLE, "|CMD")	HANDLE	Geerbt	Geerbt	"	$?
open(HANDLE, "|-", "CMD")	HANDLE	Geerbt	Geerbt	"	$?
fork + exec	Geerbt	Geerbt	Geerbt	wait/waitpid	RW

* CMD ist ein beliebiges Kommando (z.B. auch ein weiteres Perl-Skript)
  das genau so auch auf der Kommandozeile absetzbar wÃ¤re.

* Das externe Kommando CMD ist in Form eines Strings oder einer String-Variablen
  anzugeben und muss den Anforderungen des jeweiligen Betriebssystems gehorchen:
  + Kommando-Suchpfad PATH
  + Kommandozeilen-Optionen
  + Kommandozeilen-Parameter
  + Pfadnamen
    - GROSS/kleinschreibung
    - Verzeichnistrenner / oder \
    - Extensions
  + Zugriffsrechte
  + AusfÃ¼hrungsrechte
  + Quotierung von Sonderzeichen mit "..." und '...' (z.B. Leerzeichen)
  + Umlenkung der Standard-KanÃ¤le mit > >> 2> < |

* Es empfiehlt sich eine Quotierung des Kommandos per qq{CMD...}, um im
  Kommando CMD Variablen-Substitution sowie GÃ¤nsefÃ¼Ã‚M-^_chen + Hochkommas nutzen
  zu kÃ¶nnen.

```
  qq{DIR /d "C:\\Program Files (x86)"}   # Windows
  qq{ls -l / "***"                   }   # Linux
```

* Alle externen Kommandos CMD geben einen Exit-Status im Bereich 0..255 zurÃ¼ck:

```
  0            # Wenn alles in Ordnung war
  ungleich 0   # Wenn ein Fehler auftrat (beschreibt evtl. Fehlerart)
```

  Er kann nach dem Ende des Kommandos Ã¼ber die Perl-interne Variable $?
  abgefragt werden per:

```
  $exit_status = int($? / 256)
```

2) Aufruf-Verhalten

* system("CMD") ruft das externe Kommando CMD parallel auf und wartet auf sein
  Ende. Danach geht es im Perl-Skript weiter.

```
    system("DIR C:\\") or die "nicht aufrufbar ($!)";   # Windows
    system("ls -l /") or die "nicht aufrufbar ($!)";    # Linux
    print "Exit-Status: ", int($? / 256), "\n";

* system("CMD &") ruft das externe Kommando CMD parallel im Hintergrund auf und
  wartet nicht auf sein Ende, sondern fÃ¤hrt mit der Abarbeitung des Perl-Skripts
  fort. Der zurÃ¼ckgegebene Exit-Status ist immer 0.

    system("DIR C:\\ &");   # Windows
    system("ls -l / &");    # Linux
    print "Exit-Status: ", int($? / 256), "\n";   # Immer 0!

* exec("CMD") ruft das externe Kommando CMD auf und ersetzt das Perl-Skript
  damit. Das Perl-Skript wird ab dieser Stelle nicht mehr fortgesetzt
  (sofern das Kommando CMD ausfÃ¼hrbar war).

    exec("DIR C:\\") or die "nicht aufrufbar ($!)";   # Windows
    exec("ls -l /") or die "nicht aufrufbar ($!)";    # Linux
    print "Exit-Status: ", int($? / 256), "\n";       # Immer 0!

* Die Kommando-Substitution qx/CMD/ bzw. `CMD` fÃ¼hrt das externe Kommando CMD
  parallel aus und fÃ¤ngt die von ihm auf STDOUT ausgegebenen Ausgaben ab. Diese
  kÃ¶nnen als Skalar/Array im aufrufenden Perl-Skript weiterverarbeitet werden.

    $erg = qx/ls -l \//;   # Linux: Skalarer Kontext --> Eine Zeile
    @erg = qx/ls -l \//;   # Linux: Listenkontext    --> Mehrere Zeilen
    $erg = `ls -l \//`;    # Linux: Skalarer Kontext --> Eine Zeile
    @erg = `ls -l \//`;    # Linux: Listenkontext    --> Mehrere Zeilen
    $erg = qx/DIR C:\\/;   # Windows: Skalarer Kontext --> Eine Zeile
    @erg = qx/DIR C:\\/;   # Windows: Listenkontext    --> Mehrere Zeilen
    $erg = `DIR C:\\/`;    # Windows: Skalarer Kontext --> Eine Zeile
    @erg = `DIR C:\\/`;    # Windows: Listenkontext    --> Mehrere Zeilen
    print "Exit-Status: ", int($? / 256), "\n";

* open(HANDLE, "CMD|") oder open(HANDLE, "-|", "CMD") fÃ¼hrt das externe Kommando
  CMD parallel aus und erlaubt das Lesen von Daten von seiner Standard-Ausgabe
  per <HANDLE> im Perl-Skript. close(HANDLE) beendet das externe Kommando.

    open(HANDLE, "CMD|") or die "nicht ausfÃ¼hrbar ($!)";        # Variante A
    open(HANDLE, "-|", "CMD") or die "nicht ausfÃ¼hrbar ($!)";   # Variante B
      $erg = <HANDLE>;
      @erg = <HANDLE>;
      ...
    close(HANDLE);
    print "Exit-Status: ", int($? / 256), "\n";

* open(HANDLE, "|CMD") oder open(HANDLE, "|-", "CMD") fÃ¼hrt das externe Kommando
  CMD parallel aus und erlaubt das Senden von Daten auf seine Standard-Eingabe
  per print HANDLE ... im Perl-Skript. close(HANDLE) beendet das externe
  Kommando.

    open(HANDLE, "|CMD") or die "nicht ausfÃ¼hrbar ($!)";        # Variante A
    open(HANDLE, "|-", "CMD") or die "nicht ausfÃ¼hrbar ($!)";   # Variante B
      print HANDLE ...;
      print HANDLE ...;
      ...
    close(HANDLE);
    print "Exit-Status: ", int($? / 256), "\n";

* fork() dupliziert das Perl-Skript und erzeugt einen parallel laufenden
  identischen Kind-Prozess. Dieser wird oft per exec("CMD") durch ein beliebiges
  externes Kommando CMD ersetzt. Das ursprÃ¼nglich Perl-Skript ist der
  Eltern-Prozess des neu erzeugten Kind-Prozesses und wartet Ã¼blicherweise
  per "wait" oder "waitpid" auf das Ende des Kind-Prozesses.

    defined(my $pid = fork) or
      die "Kein fork mÃ¶glich: $!";
    if ($pid == 0) {      # Kindprozess
      exec "date";        # Kommando ausfÃ¼hren (Linux)
      exec "DATE /T";     # Kommando ausfÃ¼hren (Windows)
      die "Kein exec mÃ¶glich: $!";
    }
    else {                # Elternprozess
      waitpid($pid, 0);   # Auf Kindprozess warten
      print "Exit-Status: ", int($? / 256), "\n";
    }

Hinweise:

* Ist das Kommando CMD nicht ausfÃ¼hrbar, so steht die zugehÃ¶rige Fehlermeldung
  anschlieÃM-^_end in der Variablen "$!".
```

* Unter Windows sind die Quotierungszeichen '...' nicht auf der Kommandozeile
  einsetzbar. Dort funktioniert nur "...".

* Windows benÃ¶tigt den "\" als Verzeichnistrenner. Dieser ist in Perl in
  "...\\..." zu verdoppeln oder in '...' bzw. q/.../ zu setzen.