

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 1.7 2015/11/22 21:28:27 tsbirn Exp \$

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

## INHALTSVERZEICHNIS

- 1) Aufruf-Arten
- 2) Aufruf-Verhalten

### 1) Aufruf-Arten

Für den Aufruf eines externen Kommandos CMD in einem Perl-Skript gibt es folgende Möglichkeiten, je nach Anwendungszweck ist eine geeignete auszuwählen:

| 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 wäre.

\* 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
- + Ausführungsrechte
- + Quotierung von Sonderzeichen mit "... " und '...' (z.B. Leerzeichen)
- + Umlenkung der Standard-Kanäle mit > >> 2> < |

\* Es empfiehlt sich eine Quotierung des Kommandos per qq{CMD...}, um im Kommando CMD Variablen-Substitution sowie Gänsefüßchen + Hochkommas nutzen zu können.

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

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

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

Er kann nach dem Ende des Kommandos über 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 fährt mit der Abarbeitung des Perl-Skripts
fort. Der zurückgegebene 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 ausführbar 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` führt das externe Kommando CMD
parallel aus und fängt die von ihm auf STDOUT ausgegebenen Ausgaben ab. Diese
können 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") führt 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 ausführbar ($)"; # Variante A
open(HANDLE, "-|", "CMD") or die "nicht ausführbar ($)"; # Variante B
$erg = <HANDLE>;
@erg = <HANDLE>;
...
close(HANDLE);
print "Exit-Status: ", int($? / 256), "\n";

* open(HANDLE, "|CMD") oder open(HANDLE, "|-", "CMD") führt 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 ausführbar ($)"; # Variante A
open(HANDLE, "|-", "CMD") or die "nicht ausführbar ($)"; # 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 ursprünglich Perl-Skript ist der
Eltern-Prozess des neu erzeugten Kind-Prozesses und wartet üblicherweise
per "wait" oder "waitpid" auf das Ende des Kind-Prozesses.

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

```

Hinweise:

\* Ist das Kommando CMD nicht ausführbar, so steht die zugehörige Fehlermeldung anschließend in der Variablen "\$!".

- \* Unter Windows sind die Quotierungszeichen '...' nicht auf der Kommandozeile einsetzbar. Dort funktioniert nur "...".
- \* Windows benötigt den "\" als Verzeichnistrenner. Dieser ist in Perl in "...\\..." zu verdoppeln oder in '...' bzw. q/.../ zu setzen.