HOWTO zur Shee-Bang Zeile in Skript-Dateien

(C) 2006-2024 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
OSTC Open Source Training and Consulting GmbH
http://www.ostc.de

$Id: shell-shee-bang-HOWTO.txt,v 1.7 2025/02/18 10:08:31 tsbirn Exp $

Dieses Dokument beschreibt die Bedeutung der Shee-Bang Zeile in ausführbaren Dateien bzw. Skripten.

INHALTSVERZEICHNIS

1) Einführung
2) Syntax
3) Ablauf
4) Fehler
5) Beispiele

---

1) Einführung

Die Shee-Bang Zeile bestimmt, mit welchem Programm (Interpreter) eine Skript-Datei ausgeführt wird. Fehlt diese Zeile, wird die Skript-Datei automatisch mit der AKTUELLEN Shell ausgeführt, in der das Skript als Kommando aufgerufen wurde (d.h. unter Linux normalerweise mit der "bash").

Der Einsatz des Shee-Bang Konzepts ist nicht auf Shell-Skripte beschränkt, sondern kann auf jede Art von Skriptsprache angewendet werden. Neben den klassischen Shells "sh", "csh", "tcsh", "ksh", "bash" und "zsh" gibt es noch viele andere Skript-Programmiersprachen (z.B. "sed", "awk", "perl", "php", "python", "ruby", "lua", "tcl") die eine Skript-Datei ausführen können.

Ebenso können auch Kommandos als Shee-Bang Zeile eingesetzt werden, die ihre Eingabe von einer Datei lesen können (z.B. "grep", "telnet", "ftp").

2) Syntax

In einer Skript-Datei sind in der 1. Zeile + 1. Spalte die beiden Zeichen "#!" (umgangssprachlich "Shee" und "Bang") und danach der ABSOLUTE (vollständige) Pfad zum Programm anzugeben, mit dem das Skript in der Skript-Datei ausgeführt werden soll:

```
+----------------+
|#!/bin/bash     |
|...             |
+----------------+
```

Die Länge der Shee-Bang Zeile ist beschränkt (32, 64 oder 128 Zeichen je nach Linux-Version), weiterhin darf sie durch Leerraum getrennt maximal EIN zusätzliches Argument nach dem Programmpfad enthalten (hier "-u"):

```
+----------------+
|#!/bin/bash -u  |
|...             |
+----------------+
```

Außer zur Trennung von Programmpfad und evtl. vorhandenem Argument darf die Shee-Bang Zeile keine Leerzeichen oder sonstigen Zeichen enthalten und muss wie unter Linux üblich mit einem "\n" abgeschlossen sein. Insbesondere andere Zeilenende-Formate (z.B. "\r\n" bei Windows und "\r" bei MacOS X) sind nicht erlaubt und führen zu Fehlern (Pfad wird nicht gefunden):

```
  -bash: ./test.sh: /bin/bash^M: Defekter Interpreter: No such file or directory
```

Sobald eine Skript-Datei ausführbar gemacht wurde, kann sie über ihrem Namen (inklusive Extension) als Kommando aufgerufen werden.

```
  chmod a+x skript.sh   # Ausführbar machen
  ./skript.sh           # Aufrufen
```

Eine Shee-Bang Zeile, die nicht in der 1. Zeile + 1. Spalte beginnt, wird vom Linux-Kernel ignoriert. Der von der Shee-Bang Zeile ausgewälte Interpreter ignoriert die Shee-Bang Zeile grundsätzlich, weil sie für ihn als Kommentarzeile zählt (beginnt mit "#" bis zum Zeilenende).

3) Ablauf

Der Linux-Kern liest den Inhalt der ausführbaren Skript-Datei nach ihrem Aufruf ein, erkennt die beiden Zeichen "#!" als ERSTE Zeichen der Datei und interpretiert daher den Rest der 1. Zeile als absoluten Pfad zu einem Kommando, das er startet und ihm dabei die GESAMTE zum Aufruf verwendete Kommandozeile als Argument mitgibt:

```
  /bin/bash ./skript.sh
  <-------> <--------->
  Shee-Bang  Kmdozeile
```

Hierdurch wird eine (Sub)Shell "/bin/bash" gestartet, die das Skript "test.sh" ausführt. Die ursprüngliche Shee-Bang Zeile --- die diesen Ablauf auslöst --- wird dabei vom Skript-Interpreter (hier "bash") als KOMMENTARZEILE interpretiert und überlesen, d.h. ignoriert. Dies ist der Grund, warum ALLE Skriptsprachen unter Linux das Zeichen "#" als Beginn einer Kommentarzeile interpretieren.

Enthält die Shee-Bang Zeile EIN zusätzliches Argument, dann wird dieses vom Linux-Kern mit in den generierten Aufruf eingesetzt:

```
  vim test2.sh
  +--------------------+
  |#!/usr/bin/bash -ivu |
  |...                 |
  +--------------------+
  chmod +x test2.sh
  ./test2.sh
```

Obiger Ablauf führt zur Ausführung des folgenden Kommandos:

```
  /usr/bin/bash -ivu ./test2.sh
  <---------------> <-------->
      Shee-Bang       Kmdozeile
```

HINWEIS: Der Linux-Kern kopiert alle Zeichen der Shee-Bang Zeile direkt (ohne Interpretation durch eine Shell), d.h. es bedarf keiner Quotierung mit "...", '...' und "\", um Sonderzeichen zu schützen.

4) Fehler

Bei ungültigem Pfad (z.B. führender "/" vergessen) erfolgt beim Aufruf des Skripts eine Fehlermeldung der ausführenden Shell (hier der bash):

```
  bash: ./fehler.sh: bin/sh: bad interpreter: file or directory not found
```

Beispiele für ungültige Pfade in der Shee-Bang Zeile:

```
  !#/bin/sh          # # und ! vertauscht
```

```
  #/bin/sh           # ! vergessen
  # !/bin/sh         # Leerzeichen zwischen # und !
  #!bin/sh           # Pfad "bin/sh" gibt es nicht
  #!/bin/sh^M        # Strg-M (DOS/Windows-Zeilenende) im Pfad
  #!/bin/xyzabc123   # Kommando "xyzabc123" gibt es nicht


5) Beispiele
------------
a) Ein (klassisches) Bourne-Shell-Skript "skript.sh" beginnt mit "#!/bin/sh":

  more skript.sh
  +--------------------+
  |#!/bin/sh           |
  |echo "Hallo Welt!"  |
  +--------------------+
  chmod +x skript.sh
  ./skript.sh                  # --> /bin/sh ./skript.sh

b) Ein Z-Shell-Skript "zsh.sh" beginnt mit "#!/usr/bin/zsh":

  more zsh.sh
  +--------------------+
  |#!/usr/bin/zsh      |
  |echo "Hallo Welt!"  |
  +--------------------+
  chmod +x zsh.sh
  ./zsh.sh                     # --> /usr/bin/zsh ./skript.sh

c) Ein Perl-Skript "hello.pl" beginnt mit "#!/usr/bin/perl -w":

  more hello.pl
  +-------------------------+
  |#!/usr/bin/perl -w       |   # -w schaltet Warnungen ein
  |print "Hallo Welt!\n";   |
  +-------------------------+
  chmod +x hello.pl
  ./hello.pl                   # --> /usr/bin/perl -w ./hello.pl

d) Ein Grep-Skript "made.grep" beginnt mit "#!/bin/grep REGEXP", ein Kommando
   das den Inhalt des Skripts nach Zeilen durchsucht, die zum regulären Ausdruck
   "REGEXP" passen. ACHTUNG: keine Anführungszeichen um den Suchtext setzen, da
   diese sonst mitgesucht werden. Die Shell sieht diese Kommandozeile nämlich gar
   nicht, sondern nur der Linux-Kern!:

  more made.grep
  +-----------------------------------+
  |#!/bin/grep made                   |
  |echo "Dies ist ein Text mit made"  |
  |ohne ... ohne                      |
  |mit made mit made mit made         |
  |ohne ... ohne                      |
  +-----------------------------------+
  chmod +x made.grep
  ./made.grep                  # --> /bin/made.grep made ./made.grep

e) Ein Sed-Skript "vokal.sed" beginnt mit "#!/bin/sed -f" (sed = stream editor,
   programmierbarer Editor zum automatisierten Bearbeiten von Dateien). Die
   Option "-f" (file) ist notwendig, damit der sed-Interpreter das Skript aus
   einer Datei liest.

  more vokal.sed
  +-------------------------------------------------------------+
  |#!/bin/sed -f                                                |
  |# Ersetzt alle Vokale durch nichts (s=substitute, g=global) |
  |s/[aeiou]//g                                                 |
```

```
|s/[AEIOU]//g                                                    |
+---------------------------------------------------------------+
chmod +x vokal.sed
vokal.sed < INPUT > OUTPUT    # --> /bin/sed -f ./vokal.sed < INPUT > OUTPUT
```

f) Ein Awk-Skript "wc.awk" beginnt mit "#!/usr/bin/awk -f" (awk = Vorläufer von Perl). Der Pfad zum Awk-Interpreter kann unterschiedlich lauten (z.B. auch "/usr/local/bin/gawk"). Die Option "-f" (file) ist notwendig, damit der Awk-Interpreter das Skript aus einer Datei liest.

```
more wc.awk
+---------------------------------------------------------------+
|#!/usr/bin/awk -f                                              |
|# (programmiert den Linux-Befehl "wc" (word count) nach)       |
|# Zählt Zeilen, Worte und Zeichen einer Datei                  |
|{                                                              |
|    words += NF                                                |
|    chars += length($0)                                        |
|}                                                              |
|END {                                                          |
|    print NR, words, chars+NR                                  |
|}                                                              |
+---------------------------------------------------------------+
chmod +x wc.awk
./wc.awk INPUT...             # --> /usr/bin/awk -f ./wc.awk INPUT...
```

g) Ein Python-Skript "wc.py" beginnt mit "#!/usr/bin/python". Der Pfad zum Python-Interpreter kann unterschiedlich lauten (z.B. auch "/usr/local/bin/python3.7").

```
more wc.py
+---------------------------------------------------------------+
|#!/usr/bin/python                                              |
|# (programmiert den Linux-Befehl "wc" (word count) nach)       |
|# Zählt Zeilen, Worte und Zeichen einer Datei                  |
|import sys                                                     |
|                                                               |
|lines = words = chars = 0                                      |
|for (lines, zeile) in enumerate(sys.stdin, 1):                 |
|    words += len(zeile.split())                                |
|    chars += len(zeile)                                        |
|                                                               |
|print(lines, words, chars)                                     |
+---------------------------------------------------------------+
chmod +x wc.py
./wc.py < INPUT               # --> /usr/bin/python ./wc.py < INPUT
```