Probleme beim Umstieg von Awk auf Perl

© Thomas Birnthaler, OSTC GmbH
eMail: tb@ostc.de

17.7.2003 — V1.7 [awk2perl.txt]

perl (*practical extraction and reporting language*) ist eine Zusammenfassung der UNIX-Kommandos sh, sed, awk, tr, grep, sort und bietet zusätzlich noch extrem ausdrucksstarke Reguläre Ausdrücke, C-Operatoren, Zugriff auf C-Bibliotheksfunktionen und UNIX-Systemaufrufe, Objektorientierung, Modularisierung, Sicherheitsaspekte, usw. Diese Programmiersprache ist daher sehr mächtig und sehr umfangreich und ersetzt in zunehmenden Maße die klassischen UNIX-Skript-Sprachen shell, sed, awk und sogar C. Es gibt noch andere ähnlich leistungsfähige Skript-Sprachen wie z.B. PHP, Python, Ruby, Tcl/Tk, Lua.

In dieser Übersicht werden die wichtigsten Unterschiede zwischen Awk und Perl aufgelistet, um den Umstieg auf Perl zu erleichtern.

- Unterschiede in der Syntax
  - ▷ Kennt die Schreibweise MUSTER { AKTION } nicht, statt dessen MUSTER && AKTION oder AKTION if MUSTER oder if (MUSTER) { AKTION } verwenden.
  - ▷ Statt continue ist in Schleifen next zu verwenden.
  - ▷ Zu next (nächsten Eingabesatz lesen und von vorne beginnen) gibt es keine direkte Entsprechung (prinzipiell <STDIN>).
  - ▷ Zwischen Befehlen ist ein ; (Strichpunkt) notwendig (bis auf den letzten Befehl in einem Block, Zeilenenden werden nicht als Befehls-Trenner akzeptiert).
  - ▷ {...} ist in Bedingungen und Schleifen auch bei nur einer Anweisung nötig.
  - ▷ Bei skalaren Variablen ist der Präfix $, bei Arrays ist der Präfix @ anzugeben (numerischer Index 0...n). Bei Hashes (entsprechen den Awk-Arrays) ist der Präfix %, vor Funktionsnamen ist & anzugeben.
  - ▷ sub statt function vor Funktionsdefinitionen
  - ▷ open(FILE, "| sort | uniq -c > xxx") + print FILE ... statt\ print ... | "sort | uniq -c > xxx".
  - ▷ Kein Komma zwischen FILEHANDLE und 1. Argument in print angeben (zwischen den Argumenten schon).
  - ▷ & ist vor Funktionsnamen notwendig, bei Referenzen für Signal-Handler oder Aufruf ohne Argumente oder Aufruf vor Definition (wenn die Funktion noch nicht bekannt ist).

- ▷ `Strg-M` in Skripten am Zeilenende ist nicht erlaubt (MS-DOS-Format, auch bei Awk nicht).

- Unterschiede bei den Operatoren
  - ▷ Statt `~` wird `=~` verwendet (`!~` bleibt gleich, `~` bedeutet bitweises Komplement)
  - ▷ Zeichenkettenkonkatenation erfolgt durch den Operator Punkt (`"abc" . "def"`), nicht mehr durch einfaches Hintereinanderschreiben (`"abc" "def"`),
  - ▷ `eq/ne/gt/ge/lt/le` machen Zeichenkettenvergleiche, `=/!=/>/>=/</<=` machen Zahlenvergleiche (bei der Shell genau umgekehrt!). Im Awk gibt es nur die 1. Variante, die Unterscheidung erfolgt dort implizit durch den Typ der beiden Variablen.
  - ▷ Es gibt neue Operatoren `<=>` und `cmp` zum Sortieren von Zahlen/Strings.

- Unterschiede bei den Standardvariablen
  - ▷ `$_` statt `$0` enthält die aktuelle Eingabezeile.
  - ▷ `@F` statt `$1..$NF` enthält die in Worte zerlegte Eingabezeile.
  - ▷ `$0` ist der Name der Perldatei (nicht der Inhalt der aktuellen Eingabezeile).
  - ▷ `@_` enthält die Parameter von Funktionen als Referenzen (es gibt keine benannten Parameter).
  - ▷ Statt `FNR` die Variable `$.` verwenden.
  - ▷ Statt `FS` die Option `-F` verwenden.
  - ▷ Statt `RS` die Variable `$/` verwenden.
  - ▷ Statt `OFS` die Variable `$,` verwenden.
  - ▷ Statt `ORS` die Variable `$\` verwenden.
  - ▷ Statt `SEP` die Variable `$;` verwenden.
  - ▷ Statt `IGNORECASE` die Option `i` bei `m/.../` und `s/.../.../` verwenden.
  - ▷ Statt `ARGV` die Variable `@ARGV` verwenden, `ARGC` gibt es nicht (entspricht @ARGV im skalaren Kontext).

- Abweichungen bei den Standardfunktionen
  - ▷ Statt `match()` den Operator `m/.../` verwenden.
  - ▷ Statt `sub()` den Operator `s/.../.../` verwenden.
  - ▷ Statt `gsub()` den Operator `s/.../.../g` verwenden.

- Unterschiedliches Verhalten
  - ▷ Bei Perl-Kommandos direkt auf der Kommandzeile ist `-e` (execute) nötig, bei Angabe eines Skriptes auf der Kommandozeile ist `-f` (file) wegzulassen.
  - ▷ Die automatische Einleseschleife erhält man mit den Optionen `-n` (noprint) oder `-p` (print).
  - ▷ Automatischer Split der Eingabezeile erfolgt nur bei Option `-a` (+ `-n/-p`) in die Variable `@F`.

- ▷ `\n` bleibt beim Einlesen erhalten, kann mit `cho(m)p`-Funktion entfernt werden.
- ▷ `\n` wird bei `print`/`printf` nicht automatisch erzeugt, immer angeben.
- ▷ In Zeichenketten der Form `"..."` erfolgt Variableninterpolation `"...$VAR..."` und Kommandosubstitution `"...`CMD`..."`, in Zeichenketten der Form `'...'` erfolgt sie nicht.
- ▷ `,` in `print` erzeugt kein Leerzeichen.

- Unterschied bei Regulären Ausdrücken
  - ▷ In regulären Ausdrücken sind `( )` und `{ }` Metazeichen und daher zu quoten
  - ▷ `(...)` speichert Komponenten eines regulären Ausdrucks in `\1`..`\9`. Danach ist `$1`,... für die gespeicherten Komponenten verwendbar.

- Gleiches Verhalten
  - ▷ Implizierte Typkonvertierung zwischen Zahlen und Zeichenketten

- Erweiterungen
  - ▷ Kann Binärdaten verarbeiten
  - ▷ Boolescher Wert 'Falsch': `0 "" "0" () {} undef`
  - ▷ Boolscher Wert 'Wahr': Alles andere (insbesondere `"00"`)

— END —