AWK – eine fast vergessene Programmiersprache

Version 1.4, 23.6.2003

© Thomas Birnthaler, OSTC GmbH

Web: **www.ostc.de**
eMail: **tb@ostc.de**

Inhalt

UNIX

- UNIX kennt viele **Tools** (**Werkzeuge**) zur Datenverarbeitung
  - `head`, `tail`, `wc`, `cut`, `paste`, `grep`, `sed`, `sort`, `uniq`, `tr`, `diff`, `tee`, `split`, `join`, ...
  - Durch Kombination per **Pipes** oder **Shell-Skripte** entstehen daraus leistungsfähige Anwendungen
- Grundvoraussetzungen
  - **Standard-Kanäle** für Ein/Ausgabe verwendet
  - **Zeilenorientierte ASCII-Daten**
  - Keine **Programm-Meldungen**

Skript-Sprachen

- AWK ist eine **Skript-Sprache** wie die UNIX-Shell, hat aber ein anderes Prinzip:
  - Shell = "**Glue-Language**"
    - **Kombiniert UNIX-Werkzeuge** zu Programmen
    - Kann selbst nicht sehr viel
  - AWK = **vollständige Programmier-Sprache**
    - Benötigt kaum andere UNIX-Tools

Skript-Sprachen

Typische Merkmale von Skript-Sprachen

- Führen ASCII-Programmtext aus
- Keine Variablen-Deklaration
- **Wenige Datentypen**
- Automatische **Speicherverwaltung**
- Schneller **Edit-Test-Zyklus**
- Langsame Ausführungsgeschwindigkeit

Skript-Sprachen

- Unterschied **Programm** ⇔ **Skript**
  - **Programm**: 1-mal durch **Compiler** übersetzt
  - **Skript**: Jedesmal vom **Interpreter** übersetzt
- AWK geht **Zwischenweg**
  - Skript wird bei jedem Aufruf in **internen Byte-Code** übersetzt (dauert etwas), der schnell ausgeführt wird

Die Aufrufe unterscheiden sich nicht!

Skript-Sprachen

- **Vorteile** der Übersetzung in **Byte-Code**
  - Keine **Syntaxfehler** zur Laufzeit mehr möglich
  - Konstanten, Kommentare, Leerraum und Formatierung **verlangsamen** Programm **nicht**
  - **Wiederholt ausgeführter Code** wird nur 1× übersetzt und dann sehr schnell ausgeführt
- **Nachteil**
  - Bei **kleinen Programmen** überwiegt die Übersetzungszeit die Laufzeit bei weitem

Inhalt

AWK - Eigenschaften

AWK ist ein „**programmierbarer**“ **Filter**

- Verhalten wird durch **Kommandos** festgelegt
- Liest von **Standard-Eingabe**
  (oder den angegebenen Dateien)
- Schreibt auf **Standard-Ausgabe**
- Verändert verarbeitete Dateien **nicht**

AWK - Eigenschaften

Prinzipien des AWK

- AWK = Aho + Weinberger + Kernighan
- Fasst Fähigkeiten vieler **UNIX-Tools** zusammen
- Erledigt vieles **automatisch**
- **Einfache** Struktur (45 Seiten Spezifikation)
- **Reguläre Ausdrücke**
- Vollständige Programmiersprache
- Verwandtschaft mit C

AWK - Eigenschaften

Typische Anwendungen des AWK

- Dateien automatisch und fehlerfrei **bearbeiten**
- **Umformatieren** von Texten
- **Konvertieren** von Daten
- **Textverarbeitung** im weitesten Sinne
- **Prototyping** von Programmen
- **Realisierung** von Programmen

AWK - Eigenschaften

AWK-Varianten

- `oawk` – Erste Version ab 1977 (**old AWK**)
- `nawk` – Erweiterte Version ab 1985 (**new AWK**)
- `mawk` – Erweiterte Version von Michael Brennan
- `gawk` – **GNU-AWK** (schneller, bessere Fehlermeldungen, einige Erweiterungen)
- `awka` – AWK-Compiler (awka.sourceforge.net)

AWK - Eigenschaften

AWK-Varianten

- Der **Programm-Name** awk ist meist ein **symbolischer Link**
  - Normalerweise auf nawk
  - Unter Linux natürlich auf gawk ☺

AWK - Eigenschaften

Vorteile des AWK

- Leicht zu **erlernen**
- Viele **Automatismen** (Leseschleife, Zerlegung in Worte, Datentyp-Konvertierung, Speicherverwaltung)
- Schnelle, interaktive Entwicklung
- Variablen müssen **nicht deklariert** werden
- **C-ähnlich**

AWK - Eigenschaften

Nachteile des AWK

- Für grosse Datenmengen zu **langsam**
- Verarbeitung von **Binärdaten** nicht möglich
- Dateizugriff ausschließlich **sequenziell**
- Variablen müssen **nicht deklariert** werden
- Zugriff auf **Einzelzeichen** umständlich und langsam

Inhalt

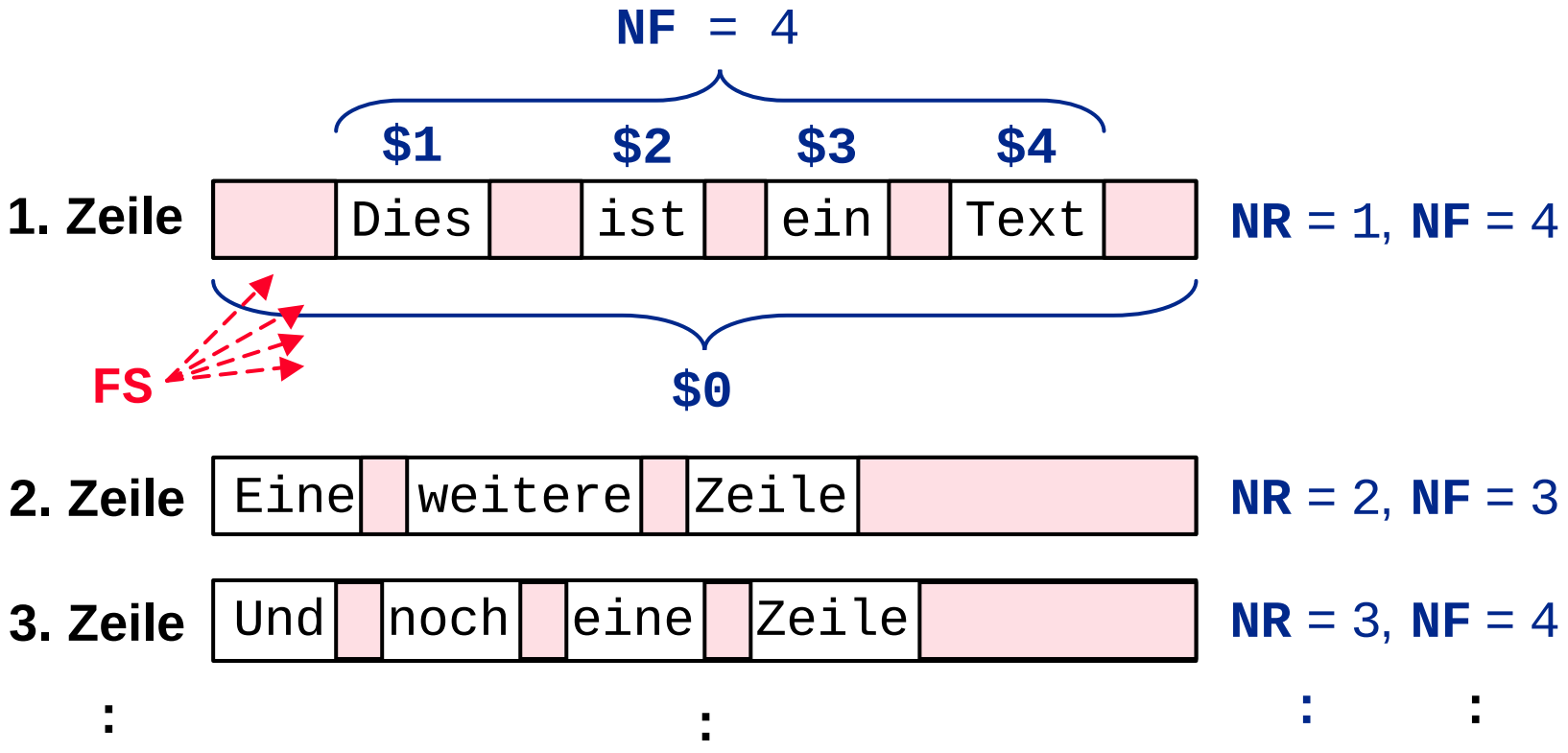
AWK – Funktionsweise

Funktionsweise des AWK

- Eingabe wird **Zeile für Zeile** gelesen
  - `$0` enthält **aktuell gelesene Zeile**
  - `NR` enthält **Zeilennummer** (**number of records**)
- Jede Zeile wird gemäß einem wählbaren Feldtrenner automatisch in **Felder** zerlegt
  - FS definiert den Feldtrenner (**field separator**)
  - Über **Variablen** $1, $2, ... ansprechbar
  - `NF` zählt **Anzahl Felder** (**number of fields**)
  - Leerraum ist **Standard-Feldtrenner**

AWK – Funktionsweise

Automatische Zeilenzerlegung des AWK



AWK – Funktionsweise

- AWK-Skripte enthalten **Muster-Aktions-Paare**

  ```
  MUSTER { AKTION }
  ```

- Wenn das **Muster** auf die aktuelle Zeile zutrifft, wird die **Aktion** ausgeführt
- Entweder *MUSTER* oder *AKTION* darf **fehlen**
  - Kein *MUSTER*
    → *AKTION* auf **alle Zeilen** anwenden
  - Keine *AKTION*
    → Zum *MUSTER* passende Zeilen **ausgeben**

AWK – Funktionsweise

- Typische **Muster**

```
NR < 10
NF != 7
$1 >= 100 && $1 <= 65500
/[Mm]ade[n]?/
```

- Typische **Aktionen**

```
{ print "Dies ist ein Text" }
{ sum = sum + $3 }
```

AWK – Funktionsweise

Beispiel

```
awk '/[Mm]ade/ { print $1 }' gedicht
```

Das erste Wort jeder Zeile ausgeben,
das den Text `made` oder `Made` enthält

```
awk '{ print $1 }' gedicht
```

Das erste Wort jeder Zeile ausgeben

```
awk '/[Mm]ade/' gedicht
```

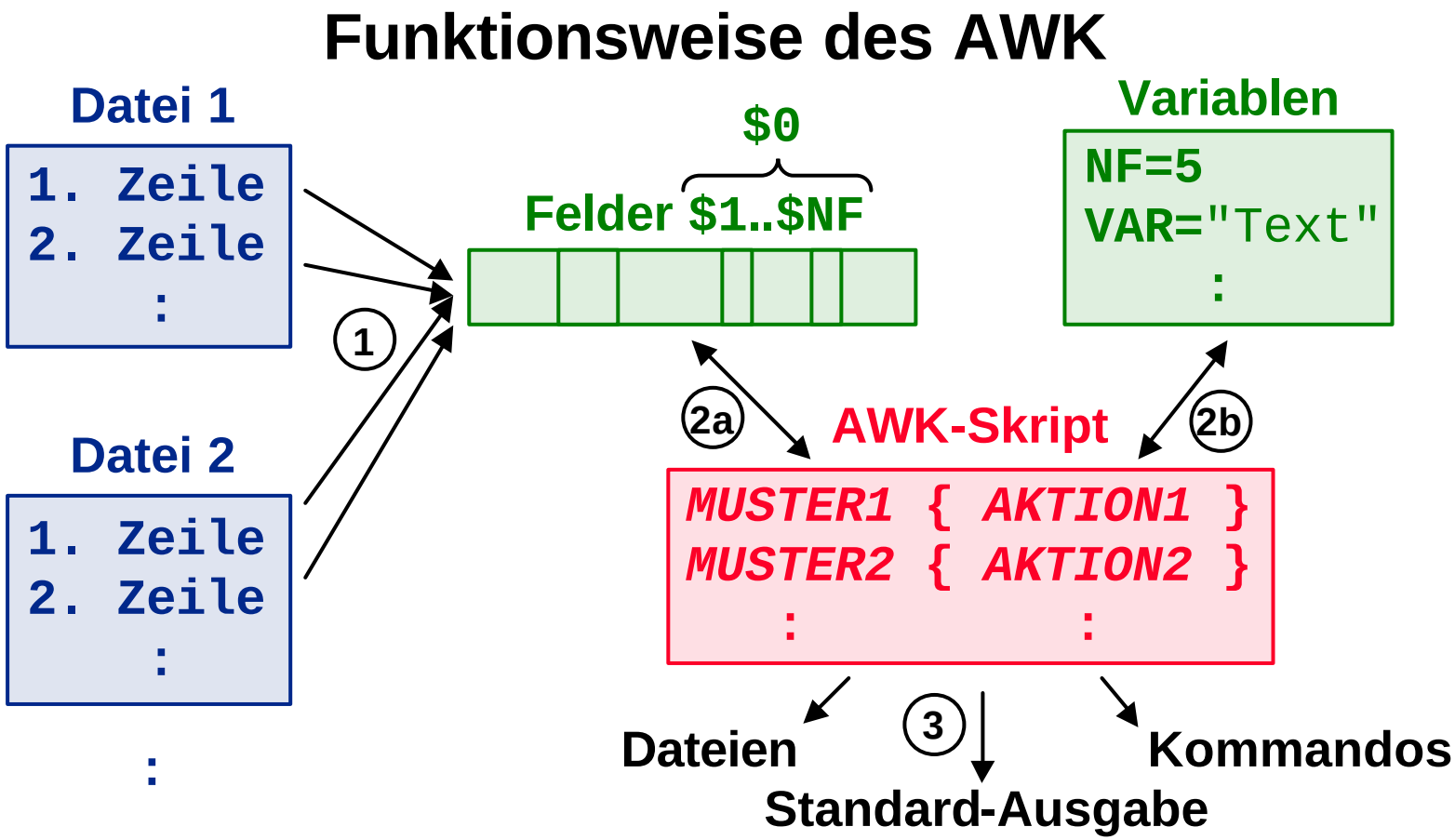
Jede Zeile ausgeben,
die Text `made` oder `Made` enthält

AWK – Funktionsweise

Automatische Leseschleife

- Angegebene Dateien Zeile für Zeile einlesen
- Die Muster werden für die aktuelle Zeile **der Reihe nach** überprüft.
  - Muster für Eingabezeile **erfüllt** → Aktion ausführen
- Nach Abarbeitung aller Muster-Aktions-Paare
  - **Nächste Zeile** lesen
  - Wieder beim ersten Muster-Aktions-Paar beginnen

AWK – Funktionsweise



AWK – Funktionsweise

Beispiel

```
awk '/[Mm][Aa][Dd][Ee]/ { print "MADE" }
     /rinde/ { print "RINDE" }' gedicht

awk 'NF < 5  { print "weniger als 5 Worte" }
     NF == 5 { print "5 Worte" }
     NF > 5  { print "mehr als 5 Worte" }' \
     gedicht
```

AWK – Funktionsweise

Optionen

`-F` *`C`*	**F**eldtrenner auf Zeichen *C* setzen
`-v` *`VAR=WERT`*	Variable *VAR* den Wert *WERT* zuweisen
`-f` *`DATEI`*	Skript *DATEI* ausführen (**f**ile)

Beispiel

```
awk -F: '{ print $1, $2 }' /etc/passwd
echo $PATH | awk -F: '{ print $1 ":" $NF }'
awk -v PRAEFIX=">> " '{ print PRAEFIX, $0 }'
awk -f wc.awk gedicht
```

Inhalt

- ✓ UNIX und Skript-Sprachen
- ✓ AWK – Eigenschaften
- ✓ AWK – Funktionsweise
- **AWK-Skripte**
- AWK – Muster, Operatoren, Kontrollstrukturen
- AWK – Variablen
- AWK – Arrays
- AWK – Funktionen
- Literatur und Links

AWK-Skripte

- AWK-Kommandos in **Skript-Datei** zusammen-fassen, falls Kommandozeile zu lang wird
- **Kommentare** durch „#“ einleiten
  - Erstrecken sich bis zum Zeilenende
  - Auch nach Kommandos erlaubt
- **Leerzeilen** werden ignoriert
- Kommandos durch "**;**" oder Return **trennbar**

  → "**;**" am Zeilenende fast immer weggelassen!

AWK-Skripte

- Skript ist über seinen Namen **aufrufbar** wenn
  - Skript-Datei **lesbar** & **ausführbar** (`rx`-Rechte)
  - **Suchpfad** `$PATH` enthält Skript-Verzeichnis
- Erste Skript-Zeile („**Shee-Bang**“-**Zeile**)

  `#!/bin/awk -f` (**file**)
  - Legt **Programm** zur Skript-Ausführung fest
  - Ohne sie wird es **von der Shell** ausgeführt

AWK-Skripte

Beispiel (*AWK-Skript* `wc.awk`)

Skript-Verzeichnis erstellen

```
mkdir awks
```

Skript erstellen (`vi awks/wc.awk`)

```
#!/bin/awk -f
{ words += NF
  chars += length($0) }
END { print NR, words, chars+NR }
```

AWK-Skripte

Beispiel (*AWK-Skript* `wc.awk`)

Ausführbar machen

```
chmod a+x awks/wc.awk
```

Suchpfad erweitern

```
PATH=$PATH:$HOME/awks
```

Aufruf

```
wc.awk gedicht
```

Startet AWK mit `wc.awk` als Skript-Datei

```
/bin/awk -f wc.awk gedicht
```

Inhalt

- ✓ UNIX und Skript-Sprachen
- ✓ AWK – Eigenschaften
- ✓ AWK – Funktionsweise
- ✓ AWK-Skripte
- **AWK – Muster, Operatoren, Kontrollstrukturen**
- AWK – Variablen
- AWK – Arrays
- AWK – Funktionen
- Literatur und Links

AWK – Muster

- **Muster-Arten**

`BEGIN`	
`END`	
Numerischer Ausdruck	(→ *Zahl*)
Logischer Ausdruck	(→ *wahr oder falsch*)
/Regulärer Ausdruck/	(*Schrägstriche*)
$Muster_1$, $Muster_2$	(**Bereichsmuster**)

AWK – Muster

- `BEGIN { Anweisungen }`
  Vor Verarbeitung der **ersten Eingabezeile**
  - Initialisieren globaler Variablen
  - Preprocessing (z.B. Kopfzeile ausgeben)
- `END { Anweisungen }`
  Nach Verarbeitung der **letzten Eingabezeile**
  - Aufräumen (Zwischendateien löschen).
  - Postprocessing (z.B. Total ausgeben)

AWK – Muster

Beispiel

```
awk 'BEGIN { print "Suche alle Maden" }
     /[Mm]ade/ { ++cnt; print ">> ", $0 }
     END { print cnt, "Maden" }'     gedicht

awk 'BEGIN { worte = 0 }      (überflüssig!)
     { worte = worte + NF }
     END { print worte }'           gedicht
```

AWK – Muster

- **Ausdrücke** verknüpfen Zahlen, Texte, Variablen, Funktions-Aufrufe und Array-Elemente über **Operatoren**

```
NF * 10 + 3          "+/-" 100 "DM"
NR - 5               sin(x) + cos(x)
++i                  arr[i] <= arr[j]
```

- Ausdruck ist „**falsch**“ bei folgenden Werten

`0`	(*Null*)
`""`	(*leere Zeichenkette*)

  – In allen anderen Fällen ist er „**wahr**“

AWK – Operatoren

- **Operatoren (I)** (*Reihenfolge = Vorrang*)

`(...)`	Klammerung
`$`	Feldzugriff ($1, $2, ..., $NF)
`++ --`	Inkrement/Dekrement
`^`	Potenzierung x^y
`+ - !`	Unäres Plus/Minus, logisch **NICHT**
`* / %`	Multiplikation, Divison, Modulo
`+ -`	Addition, Subtraktion
`Leerzeichen`	**Konkatenation** von Zeichenketten

AWK – Operatoren

- **Operatoren (II)** (*Reihenfolge = Vorrang*)

`== != < <= > >=`	Vergleich (*Zahlen + Strings!*)
`~ !~`	Vergleich mit **Regulärem Ausdruck**
`in`	Test, ob Index in Array vorkommt
`&&`	Logisch **UND** (*short-cut!*)
`||`	Logisch **ODER** (*short-cut!*)
`?:`	Bedingte Operation
`= += -= *= /= %= ^=`	Operation + Zuweisung

AWK – Operatoren

- **Vergleiche**
  - Für **Zahlen** und **Strings** möglich.
  - **Inhalt** der verglichenen Werte entscheidet, welcher **Vergleich** (Zahl oder String) durchgeführt wird.
- **Regulärer Ausdruck** in `/ . . . /` wird mit der ganzen Zeile `$0` verglichen.

„**wahr**“	= passt auf Zeile `$0`
„**falsch**“	= passt nicht auf Zeile `$0`

AWK – Operatoren

- **Muster-Erkennungsausdrücke** vergleichen beliebige Variablen mit Regulären Ausdrücken

```
$1 ~ /[Mm]ade/
EINGABE !~ /[Jj]a|[Nn]ein/
```

- **Bereichsausdrücke** sind „**wahr**“, sobald der 1. Teil auf eine Zeile passt bis der 2. Teil passt
  - Kann sich beliebig oft **wiederholen**

```
/Anton/ , /Thomas/
$3 >= 100 , $3 <= 999
```

AWK – Operatoren

Beispiel

```
awk 'NF <= 2'                    gedicht
awk -F: 'NF != 7'                /etc/passwd
awk -F: '$7 == "/bin/csh"'       /etc/passwd

awk '/[Mm]ade/ { print $2 }'           gedicht
awk '$1 !~ /[Mm]ade/ { print $2 }'     gedicht

awk 'NR == 10, NR == 15'   gedicht
awk '/Anfang/,/Ende/'      gedicht
```

AWK – Kontrollstrukturen

Kontrollstrukturen (*analog C*)

```
for (START; BEDINGUNG; INKREMENT) ...
while (BEDINGUNG) ...
do ... while (BEDINGUNG)
if (BEDINGUNG) ... [else ...]
break
continue
exit(CODE)
next          (AWK-spezifisch: Nächste Zeile lesen)
```

AWK – Kontrollstrukturen

Beispiel

```
awk –v nr=100 '{ while (nr != 1)
                    if (nr % 2 == 0)
                        nr /= 2
                    else
                        nr = 3 * nr + 1
                }'

awk '{ if (NF <= 5)
           print "weniger/gleich 5 Worte"
       else
           print "mehr als 5 Worte" }' gedicht
```

AWK – Kontrollstrukturen

Beispiel

```
awk -F: '{ for (feld = NF; feld > 1; --feld)
                printf $feld ":"
            print $1 }' /etc/passwd

awk '/made/ { next }
    /rinde/ { exit(0) }
    { ++zeile }
    END { print zeile, "Zeilen ohne made(n)",
          "bis zur rinde gefunden" }' gedicht
```

Inhalt

AWK – Variablen

Variablen

- **Datentypen**: Zeichenkette + (Fließkomma)Zahl
  - Unterscheidung erfolgt **automatisch**

> ***Idee**: Daten, die wie Zahlen aussehen, werden auch als Zahl behandelt*

- Mit 0 bzw. "" (*leere Zeichenkette*) **initialisiert**
- Viele **vordefinierte Variablen**
  - Werden **GROSS** geschrieben

AWK – Variablen

Beispiel

```
var = var + 10          (→ 10)
var = var * 200         (→ 2000)
++var                   (→ 2001)
print var               (→ 2001)

var = "abc" var "xyz"   (→ "abc2001xyz")
print var               (→ "abc2001xyz")
```

AWK – Variablen

Automatische Datentyp-Konvertierung

	String	Nummer	StrNum
String	String	String	String
Nummer	String	Nummer	Nummer
StrNum	String	Nummer	Nummer

AWK – Variablen

Beispiel

```
var1 = 10
var2 = "2abc"
print var1 "xyz"                    (→ "10xyz")
print var1 + var2                   (→ 12)

if (var1 > var2)
    print "Zahlenvergleich"         (10 > 2)
else
    print "Textvergleich"           (10 < 2)
```

AWK – Variablen

Wichtige **Standard-Variablen**

`NF`	Anzahl Felder (**number of fields**)
`NR`	Anzahl Zeilen insgesamt (**number of records**)
`FNR`	Anzahl Zeilen pro Datei (**file number of records**)
`FS`	Feldtrenner (**field separator**, " " = Leerraum)
`RS`	Zeilentrenner (**record separator**, "\n")
`OFS`	Ausgabe-Feldtrenner (**output field separator,** " ")
`ORS`	Ausgabe-Zeilentrenner (**output record sep.**, "\n")
`FILENAME`	Aktueller Dateiname
`IGNORECASE`	Gross/Kleinschreibung ignorieren

Inhalt

AWK – Arrays

Assoziative Arrays (**Hashes**)

- *ARR*[*INDEX*] steht für das **Element** eines **Arrays** *ARR* mit dem **Index** *INDEX*
  - **Beliebige Zeichenkette** als Index
  - Beliebige Inhalte (*auch Mischung aus Text und Zahl*)
  - Zugriff per **Hashing** (*extrem schnell*)
  - Arrays wachsen automatisch
  - Die Elemente liegen **unsortiert** im Speicher
  - Begriff "**Feld**" hat im Awk eine andere Bedeutung.

AWK – Arrays

Assoziative Arrays

- **Existenz** eines Arrayelements überprüfen

```
INDEX in ARR
```

- **Alle Indices** eines Arrays durchlaufen

```
for (i in ARR)
    print i, ARR[i]
```

- Array-Element oder ganzes Array **löschen**

```
delete ARR[i]
delete ARR
```

AWK – Arrays

Beispiel

```
arr[2]    = 100
arr[-3]   = "Text"
arr["abc"] = 123.456

if ("abc" in arr)
    print "abc ist Index"

for (i in arr)
    print "Element", i, "hat Wert", arr[i]
```

AWK – Arrays

Mehrdimensionale Arrays

- Indices durch **Komma** (bzw. SUBSEP) trennen
  - Werden als ein String behandelt
  - Subindices nicht einzeln aufzählbar, nur gemeinsam
- **Existenz** eines Arrayelements überprüfen

```
(i,j,k) in ARR
```

- **Alle Indices** durchlaufen

```
for ((i,j,k) in ARR)
    print i, j, k, ARR[i,j,k]
```

AWK – Arrays

Beispiel

```
arr[1,1]  = 100
arr[1,2]  = "Text"
arr[2,1]  = -1
arr[2,2]  = 3.141592

if ((2,2) in arr)
    print "(2,2) ist Index"

for ((i,j) in arr)
    print "Element", i, j, "hat Wert", arr[i,j]
```

AWK – Arrays

Wichtige **Standard-Arrays**

`ARGC`	Anzahl Kommandozeilen-Argumente (**argument count**)
`ARGV[...]`	Kommandozeilen-Argumente-Array (**argument vector**)
`ENVIRON[...]`	Umgebungsvariablen-Array (**environment**)

Inhalt

AWK – Funktionen

- Funktion **definieren**

```
function FUNCNAME(PARAM1, PARAM2, ...)
{
    ANWEISUNG
    ...
}
```

  - Variablen *PARAM1*, ... **lokal** zur Funktion
  - Beim **Aufruf** werden die Parameter **initialisiert**

AWK – Funktionen

- Funktion **aufrufen**
  - Beliebige **Ausdrücke** als Argumente *ARG1*, ...
  - Ihre Werte werden in den Parametern abgelegt

```
[VAR =] FUNCNAME(ARG1, ARG2, ...)
```

- Ergebnis **zurückgeben**

```
return WERT
```

AWK – Funktionen

Beispiel

```
function error(err_nr, err_msg)
{
    if (err_nr < 100)
        print "warning (" FNR "):", err_msg
    else {
        print "error (" FNR "):", err_msg
        exit(err_nr)
    }
}

error(99, "Division durch 0")
error(100, "Keine Datei angegeben")
```

AWK – Funktionen

Beispiel

```
function tan(x)
{
    return sin(x) / cos(x)
}

function fakultaet(x)
{
    prod = x;
    for ( ; x > 1; --x)
        prod *= x;
    return prod
}
```

AWK – Funktionen

Wichtige **Standard-Funktionen**

`print` *`LISTE`*	*`LISTE`* ausgeben
`printf(`*`FMT`*`,` *`LISTE`*`)`	Analog gemäß *`FMT`*
`length(`*`TEXT`*`)`	Länge von *`TEXT`*
`substr(`*`TEXT`*`,` *`START`*`,` *`LEN`*`)`	Teilstück von *`TEXT`*
`sub(`*`REGEX`*`,` *`TEXT`*`,` *`VAR`*`)`	*`REGEX`* in *`VAR`* durch *`TEXT`* 1× ersetzen
`gsub(`*`REGEX`*`,` *`TEXT`*`,` *`VAR`*`)`	Analog, aber global

AWK – Funktionen

Beispiel

```
print "i hat den Wert", i

printf("Gesamtbetrag: %.2f DM\n", summe)

LEN = length($0)

TXT = substr("Textstück", 4, 3)          (→ "tst")

sub(/t/, "@", "Textstück")          (→ "Tex@stück")

gsub(/t/, "@", "Textstück")         (→ "Tex@s@ück")
```

AWK – Funktionen

Mathematische **Standard-Funktionen**

`sqrt(x)`	Quadratwurzel von x
`sin(x)`	Sinus von x (in Radiant)
`cos(x)`	Cosinus von x (in Radiant)
`atan2(x,y)`	Inverser Tangens von (x / y)
`exp(x)`	Exponentiation e^x von x
`log(x)`	Natürlicher Logarithmus von x
`int(x)`	Ganzzahliger Teil von x
`rand()`	Zufallszahl x aus dem Bereich $0 \leq x < 1$

AWK – Funktionen

Beispiel

```
WURZEL = sqrt(100)                (→ 1.0)
print sin(3.141592 / 2)           (→ 1.0)
print cos(3.141592 / 2)           (→ 3.26795e-07)
E = exp(1)                        (→ 2.71828)
print log(10)                     (→ 2.30259)
print int(3.141592)               (→ 3)
ZUFALL = rand()                   (→ 0.237788)
```

Inhalt

- ✓ UNIX und Skript-Sprachen
- ✓ AWK – Eigenschaften
- ✓ AWK – Funktionsweise
- ✓ AWK-Skripte
- ✓ AWK – Muster, Operatoren, Kontrollstrukturen
- ✓ AWK – Variablen
- ✓ AWK – Arrays
- ✓ AWK – Funktionen
- **Literatur und Links**

Literatur

- **The AWK Programming Language**, *Aho, Weinberger, Kernighan*, Addison-Wesley, 1992, ISBN 0-291-07981-X (≈ 40 Euro)
  - *Der „Klassiker“* von den Autoren der Sprache, mit vielen lehrreichen Programmierbeispielen.
- **Effective AWK Programming**, *Arnold Robbins*, Specialized Systems Consultants, 1996, ISBN 0-916151-88-3, (≈ 30 Euro)
  - Umfassende Beschreibung des gawk von seinem Maintainer (GNU-AWK, POSIX-Definition).

Literatur

- **Reguläre Ausdrücke**, *Jeffrey Friedl*, O`Reilly, 1998, ISBN 3-930673-62-2 (≈ 30 Euro)
  - Erschöpfend (*im wahrsten Sinne des Wortes*), behandelt Unterschiede und Besonderheiten der Regulären Ausdrücke diversen UNIX-Werkzeuge.
- **Der UNIX Werkzeugkasten – Programmieren mit UNIX**, *Kernighan, Pike*, Hanser, 1987, ISBN 3-446-14273-8 (≈ 30 Euro)
  - Hier zeigen UNIX-Entwickler ihre kenntnisreiche Nutzung des Systems (1. Hälfte UNIX, 2. Hälfte C).

Links

www.ostc.de (Dieses Skript und weitere Infos)

awka.sourceforge.net (AWK-Compiler)

cm.bell-labs.com/cm/cs/awkbook
(The AWK Programming Language)

www.cs.hmc.edu/tech_docs/qref/awk.html
(Getting Started with awk)

www.canberra.edu.au/~sam/whp/awk-guide.html
(Guide to awk)

www.novia.net/~phridge/programming/awk
(Awk Programming examples)

Inhalt

- ✓ UNIX und Skript-Sprachen
- ✓ AWK – Eigenschaften
- ✓ AWK – Funktionsweise
- ✓ AWK-Skripte
- ✓ AWK – Muster, Operatoren, Kontrollstrukturen
- ✓ AWK – Variablen
- ✓ AWK – Arrays
- ✓ AWK – Funktionen
- ✓ Literatur und Links

AWK – Todo

- `getline`
- `exit`
- `pipe`
- `Umlenkung`
- `system`
- `next`
- `break / continue`
- 64-Bit double verwendet
- Gawk-Adresse
- Weitere Standard-Arrays