Guía Completa de AWK - De Principiante a Experto

Tabla de Contenidos

Introducción a AWK

AWK es un lenguaje de programación interpretado diseñado para el procesamiento de texto y extracción de datos. Su nombre proviene de las iniciales de sus creadores: Alfred **A**ho, Peter **W**einberger y Brian **K**ernighan.

Características Principales:

- **Orientado a líneas**: Procesa archivos línea por línea
- **Orientado a campos**: Divide automáticamente cada línea en campos
- **Expresiones regulares**: Soporte nativo para patrones complejos
- **Variables automáticas**: Muchas variables útiles predefinidas
- **Sintaxis similar a C**: Fácil de aprender si conoces C

Sintaxis Básica:

bash

```
awk 'patrón { acción }' archivo
awk -f script.awk archivo
echo "datos" | awk 'patrón { acción }'
```

Estructura Básica de AWK

Un programa AWK consiste en una serie de reglas de la forma:

```
awk

patrón { acción }
```

Componentes:

1. **Patrón**: Condición que determina cuándo ejecutar la acción
2. **Acción**: Conjunto de comandos a ejecutar
3. **Campos**: Datos separados por delimitadores (por defecto espacios/tabs)

Ejemplo Simple:

```
bash

# Imprimir todas las líneas
awk '{ print }' archivo.txt

# Imprimir solo la primera columna
awk '{ print $1 }' archivo.txt

# Imprimir líneas que contengan "error"
awk '/error/ { print }' log.txt
```

Variables Integradas

AWK proporciona muchas variables predefinidas extremadamente útiles:

Variables de Campo:

- `$0`: Línea completa actual
- `$1, $2, $3...`: Campos individuales (columnas)
- `$NF`: Último campo de la línea
- `$(NF-1)`: Penúltimo campo

Variables de Control:

- `NR`: Número de registro (línea) actual
- `FNR`: Número de registro en el archivo actual
- `NF`: Número de campos en la línea actual
- `FILENAME`: Nombre del archivo siendo procesado

Variables de Configuración:

- `FS`: Separador de campo (Field Separator)
- `OFS`: Separador de campo de salida (Output Field Separator)
- `RS`: Separador de registro (Record Separator)
- `ORS`: Separador de registro de salida (Output Record Separator)

Ejemplos Prácticos:

bash

```
# Mostrar número de línea y contenido
awk '{ print NR ": " $0 }' archivo.txt

# Mostrar número de campos por línea
awk '{ print "Línea " NR " tiene " NF " campos" }' archivo.txt

# Cambiar separador de campo
awk -F',' '{ print $1, $3 }' archivo.csv
# o
awk 'BEGIN { FS="," } { print $1, $3 }' archivo.csv
```

Patrones y Acciones

Tipos de Patrones:

1. Patrón Vacío (Sin patrón):

bash

```
awk '{ print $1 }' archivo.txt  # Ejecuta para todas las líneas
```

2. Expresiones Regulares:

bash

```
awk '/^[0-9]+$/ { print "Número: " $0 }' archivo.txt
awk '/error|warning/ { print FILENAME ":" NR ":" $0 }' log.txt
```

3. Expresiones Relacionales:

bash

```
awk '$3 > 100 { print $1, $3 }' datos.txt
awk 'NF == 5 { print "Línea completa: " $0 }' archivo.txt
awk 'length($0) > 80 { print "Línea larga: " NR }' archivo.txt
```

4. Patrones Compuestos:

bash

```
awk '$1 == "ERROR" && $2 > 1000 { print }' log.txt
awk '/inicio/ || /fin/ { print NR ": " $0 }' archivo.txt
```

5. Rangos de Patrones:

bash

```
awk '/START/,/END/ { print }' archivo.txt  # Desde START hasta END
awk 'NR==5,NR==10 { print }' archivo.txt  # Líneas 5 a 10
```

Bloques Especiales

Bloque BEGIN:

Se ejecuta **antes** de procesar cualquier línea del archivo.

bash

```
awk 'BEGIN {
    print "Iniciando procesamiento..."
    FS = ","
    OFS = " | "
    contador = 0
}
{
    contador++
    print $1, $2
}
END {
    print "Procesadas " contador " líneas"
}' archivo.csv
```

Usos del bloque BEGIN:

- Inicializar variables
- Configurar separadores (FS, OFS, RS, ORS)
- Imprimir encabezados
- Definir funciones
- Configurar formato de números

Bloque END:

Se ejecuta **después** de procesar todas las líneas.

bash

```
awk '{ suma += $1; count++ }
END {
  if (count > 0)
    print "Promedio: " suma/count
  else
    print "No hay datos"
}' numeros.txt
```

Usos del bloque END:

- Imprimir resúmenes
- Calcular totales y promedios
- Generar reportes finales
- Limpiar recursos

Ejemplo Completo con BEGIN y END:

bash

```
awk 'BEGIN {
  print "=== REPORTE DE VENTAS ==="
  print "Fecha\t\tVendedor\tMonto"
  print "================================"
  total = 0
  count = 0
}
{
  print $1 "\t" $2 "\t\t$" $3
  total += $3
  count++
}
END {
  print "================================"
  print "Total vendido: $" total
  print "Promedio por venta: $" (count > 0 ? total/count : 0)
  print "Número de ventas: " count
}' ventas.txt
```

Operadores y Expresiones

Operadores Aritméticos:

bash

```
+  # Suma
-  # Resta
*  # Multiplicación
/  # División
%  # Módulo
^  # Potencia
++ # Incremento
-- # Decremento
```

Operadores de Comparación:

bash

```
== # Igual
!= # Diferente
<  # Menor que
<= # Menor o igual
>  # Mayor que
>= # Mayor o igual
~  # Coincide con expresión regular
!~ # No coincide con expresión regular
```

Operadores Lógicos:

bash

```
&&  # AND lógico
||  # OR lógico
!   # NOT lógico
```

Ejemplos:

bash

```
# Operaciones aritméticas
awk '{ print $1 + $2, $1 * $2, $1 / $2 }' numeros.txt

# Comparaciones
awk '$1 > 50 && $2 < 100 { print "Válido: " $0 }' datos.txt

# Expresiones regulares
awk '$1 ~ /^[A-Z]/ { print "Empieza con mayúscula: " $1 }' nombres.txt
```

Funciones Integradas

Funciones de Cadena:

length(string):

bash

```
awk '{ print $1, length($1) }' archivo.txt
awk 'length($0) > 100 { print "Línea larga: " NR }' archivo.txt
```

substr(string, start, length):

bash

```
awk '{ print substr($1, 1, 3) }' archivo.txt  # Primeros 3 caracteres
awk '{ print substr($0, 5) }' archivo.txt     # Desde el 5º carácter
```

index(string, substring):

bash

```
awk '{ pos = index($0, "error"); if (pos > 0) print "Error en posición " pos }' log.txt
```

split(string, array, separator):

bash

```
awk '{
   n = split($0, palabras, " ")
   for (i = 1; i <= n; i++) {
      print i ": " palabras[i]
   }
}' archivo.txt
```

gsub(regex, replacement, target) y **sub(regex, replacement, target)**:

bash

```
# gsub reemplaza todas las ocurrencias
awk '{ gsub(/error/, "ERROR"); print }' log.txt

# sub reemplaza solo la primera ocurrencia
awk '{ sub(/^[ \t]+/, ""); print }' archivo.txt  # Elimina espacios al inicio
```

toupper(string) y **tolower(string)**:

bash

```
awk '{ print toupper($1), tolower($2) }' archivo.txt
```

Funciones Matemáticas:

`sqrt(x)`, `sin(x)`, `cos(x)`, `atan2(y,x)`:

bash

```
awk '{ print $1, sqrt($1) }' numeros.txt
```

`int(x)`, `rand()`, `srand()`:

bash

```
awk 'BEGIN { srand(); print int(rand() * 100) }'  # Número aleatorio 0-99
```

`printf` para formato:

bash

```
awk '{ printf "%-10s %8.2f\n", $1, $2 }' datos.txt
```

Arrays y Estructuras de Datos

AWK soporta arrays asociativos (hash tables) que son extremadamente útiles:

Arrays Básicos:

bash

```
awk '{
  palabras[NR] = $1
  longitudes[NR] = length($1)
}
END {
  for (i = 1; i <= NR; i++) {
    print i ": " palabras[i] " (" longitudes[i] " caracteres)"
  }
}' archivo.txt
```

Arrays Asociativos:

bash

```
# Contar ocurrencias de palabras
awk '{
  for (i = 1; i <= NF; i++) {
    count[$i]++
  }
}
END {
  for (word in count) {
    print word ": " count[word]
  }
}' archivo.txt
```

Operador `in` para Arrays:

bash

```
awk 'BEGIN {
  frutas["manzana"] = 5
  frutas["banana"] = 3
  frutas["naranja"] = 8
}
{
  if ($1 in frutas) {
    print $1 " está disponible: " frutas[$1] " unidades"
  } else {
    print $1 " no está disponible"
  }
}' consultas.txt
```

Función `delete`:

bash

```
awk '{
  datos[$1] = $2
}
END {
  delete datos["temporal"]  # Eliminar elemento específico
  # delete datos        # Eliminar todo el array

  for (key in datos) {
    print key ": " datos[key]
  }
}' archivo.txt
```

Control de Flujo

Condicionales:

`if-else`:

bash

```
awk '{
  if ($1 > 100) {
    print $1 " es mayor que 100"
  } else if ($1 > 50) {
    print $1 " está entre 50 y 100"
  } else {
    print $1 " es menor o igual a 50"
  }
}' numeros.txt
```

Operador ternario:

bash

```
awk '{ print ($1 > 50) ? "Alto" : "Bajo" }' numeros.txt
```

Bucles:

`for` tradicional:

bash

```
awk '{
  for (i = 1; i <= NF; i++) {
    print "Campo " i ": " $i
  }
}' archivo.txt
```

`for-in` para arrays:

bash

```
awk '{
  for (i = 1; i <= NF; i++) {
    words[$i]++
  }
}
END {
  for (word in words) {
    print word ": " words[word]
  }
}' archivo.txt
```

while:

```bash
awk '{
  i = 1
  while (i <= NF) {
    if (length($i) > 5) {
      print "Palabra larga: " $i
    }
    i++
  }
}' archivo.txt
```

Control de Flujo:

next: Salta al siguiente registro

```bash
awk '{
  if ($1 == "SKIP") {
    next
  }
  print $0
}' archivo.txt
```

exit: Termina el programa

```bash
awk '{
  if ($1 == "END") {
    print "Terminando procesamiento"
    exit
  }
  print $0
}' archivo.txt
```

Funciones Definidas por el Usuario

Definición de Funciones:

bash

```
awk '
function factorial(n) {
    if (n <= 1) {
        return 1
    } else {
        return n * factorial(n - 1)
    }
}

function es_primo(n) {
    if (n < 2) return 0
    for (i = 2; i * i <= n; i++) {
        if (n % i == 0) return 0
    }
    return 1
}

{
    print $1, factorial($1), es_primo($1) ? "primo" : "no primo"
}' numeros.txt
```

Funciones con Variables Locales:

bash

```
awk '
function estadisticas(arr, n,    suma, i, promedio) {
    # Parámetros formales: arr, n
    # Variables locales: suma, i, promedio (después de espacios extra)

    suma = 0
    for (i = 1; i <= n; i++) {
        suma += arr[i]
    }
    promedio = suma / n

    print "Suma: " suma
    print "Promedio: " promedio
    return promedio
}

{
    numeros[NR] = $1
}
END {
    estadisticas(numeros, NR)
}' datos.txt
```

Ejemplos Prácticos

1. Procesamiento de Logs:

bash

```bash
# Analizar log de Apache
awk '
BEGIN {
    print "=== ANÁLISIS DE LOG ==="
}
{
    ip = $1
    timestamp = $4
    request = $7
    status = $9
    size = $10

    ips[ip]++
    statuses[status]++
    total_size += size
    requests++
}
END {
    print "\n=== IPs más frecuentes ==="
    for (ip in ips) {
        print ip ": " ips[ip] " requests"
    }

    print "\n=== Códigos de estado ==="
    for (status in statuses) {
        print status ": " statuses[status]
    }

    print "\n=== Resumen ==="
    print "Total requests: " requests
    print "Promedio MB/request: " (total_size/requests/1024/1024)
}' access.log
```

2. Procesamiento de CSV:

bash

```bash
# Procesar archivo CSV de ventas
awk '
BEGIN {
    FS = ","
    OFS = " | "
    print "Vendedor | Producto | Cantidad | Precio | Total"
    print "=============================================="
}
NR > 1 {  # Saltar encabezado
    vendedor = $1
    producto = $2
    cantidad = $3
    precio = $4
    total = cantidad * precio

    print vendedor, producto, cantidad, precio, total

    ventas_vendedor[vendedor] += total
    ventas_producto[producto] += cantidad
    gran_total += total
}
END {
    print "=============================================="
    print "\nVentas por vendedor:"
    for (v in ventas_vendedor) {
        print v ": $" ventas_vendedor[v]
    }

    print "\nCantidad por producto:"
    for (p in ventas_producto) {
        print p ": " ventas_producto[p]
    }

    print "\nTotal general: $" gran_total
}' ventas.csv
```

3. Formateo de Datos:

bash

```
# Formatear tabla de datos
awk '
BEGIN {
    printf "%-15s %-10s %-12s %-8s\n", "Nombre", "Edad", "Salario", "Depto"
    printf "%-15s %-10s %-12s %-8s\n", "======", "====", "=======", "====="
}
{
    printf "%-15s %-10s $%-11.2f %-8s\n", $1, $2, $3, $4
}' empleados.txt
```

Casos de Uso Avanzados

1. Procesamiento de Archivos de Configuración:

bash

```
# Procesar archivo de configuración estilo INI
awk '
BEGIN {
    seccion = ""
}
/^\[.*\]$/ {
    seccion = substr($0, 2, length($0) - 2)
    next
}
/^[^#]/ && seccion != "" {
    if (index($0, "=") > 0) {
        split($0, parts, "=")
        key = parts[1]
        gsub(/^[ \t]+|[ \t]+$/, "", key)  # Trim espacios
        value = parts[2]
        gsub(/^[ \t]+|[ \t]+$/, "", value)

        config[seccion][key] = value
        print seccion "." key " = " value
    }
}' config.ini
```

2. Generación de Reportes HTML:

bash

```
awk '
BEGIN {
  print "<html><head><title>Reporte</title></head><body>"
  print "<h1>Reporte de Datos</h1>"
  print "<table border=\"1\">"
  print "<tr><th>ID</th><th>Nombre</th><th>Valor</th></tr>"
}
{
  print "<tr><td>" $1 "</td><td>" $2 "</td><td>" $3 "</td></tr>"
}
END {
  print "</table></body></html>"
}' datos.txt > reporte.html
```

3. Validación de Datos:

bash

```
# Validar formato de email
awk '
function es_email_valido(email) {
  return email ~ /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/
}

{
  if (es_email_valido($1)) {
    print "✓ " $1 " - Válido"
  } else {
    print "✗ " $1 " - Inválido"
    errores++
  }
}
END {
  print "\nEmails inválidos: " (errores + 0)
}' emails.txt
```

4. Análisis de Rendimiento:

bash

```bash
# Analizar tiempos de respuesta
awk '
BEGIN {
  print "=== ANÁLISIS DE RENDIMIENTO ==="
}
{
  tiempo = $3
  tiempos[NR] = tiempo
  suma += tiempo
  if (tiempo > max || max == "") max = tiempo
  if (tiempo < min || min == "") min = tiempo
}
END {
  promedio = suma / NR

  # Calcular desviación estándar
  for (i = 1; i <= NR; i++) {
    suma_cuadrados += (tiempos[i] - promedio)^2
  }
  desviacion = sqrt(suma_cuadrados / NR)

  print "Mínimo: " min " ms"
  print "Máximo: " max " ms"
  print "Promedio: " promedio " ms"
  print "Desviación estándar: " desviacion " ms"

  # Percentiles
  print "\nTiempos > promedio + 2σ (outliers):"
  for (i = 1; i <= NR; i++) {
    if (tiempos[i] > promedio + 2 * desviacion) {
      print "Línea " i ": " tiempos[i] " ms"
    }
  }
}' tiempos.txt
```

Consejos y Mejores Prácticas

1. Optimización:

- Usa `BEGIN` para inicializaciones costosas
- Evita expresiones regulares complejas en bucles internos
- Usa `next` para saltar procesamiento innecesario

2. Depuración:

bash

```
# Agregar debug
awk '
BEGIN { debug = 1 }
{
  if (debug) print "DEBUG: Procesando línea " NR ": " $0
  # ... resto del código
}' archivo.txt
```

3. Manejo de Errores:

bash

```
awk '
{
  if (NF < 3) {
    print "ERROR: Línea " NR " no tiene suficientes campos" > "/dev/stderr"
    next
  }
  # ... procesar línea normal
}' archivo.txt
```

4. Modularidad:

bash

```
# Separar lógica en funciones
awk '
function procesar_linea(datos) {
  # Lógica específica
}

function validar_datos(datos) {
  # Validaciones
}

{
  if (validar_datos($0)) {
    procesar_linea($0)
  }
}' archivo.txt
```
