AWK Command Reference Guide

Overview

AWK is a powerful pattern-scanning and processing language for text files. It processes files line by line and can perform complex text manipulation, calculations, and reporting tasks.

Basic Syntax

```bash
awk 'pattern { action }' filename
awk -f script.awk filename
```

Built-in Variables

- `NR` - Number of records (line number)
- `NF` - Number of fields in current record
- `$0` - Entire current record
- `$1, $2, $3...` - Field 1, 2, 3, etc.
- `FS` - Field separator (default: whitespace)
- `RS` - Record separator (default: newline)
- `OFS` - Output field separator (default: space)
- `ORS` - Output record separator (default: newline)
- `FILENAME` - Current filename being processed
- `FNR` - File number of records (resets for each file)

Common Patterns

- `BEGIN` - Execute before processing any records
- `END` - Execute after processing all records
- `/pattern/` - Match lines containing pattern
- `$1 == "value"` - Match when field 1 equals value
- `NR == 5` - Match line 5
- `NF > 3` - Match lines with more than 3 fields

Essential Commands

Print Operations

bash

```
# Print entire file
awk '{ print }' file.txt

# Print specific fields
awk '{ print $1, $3 }' file.txt

# Print with custom separator
awk '{ print $1 ":" $2 }' file.txt

# Print line numbers
awk '{ print NR, $0 }' file.txt

# Print lines matching pattern
awk '/pattern/ { print }' file.txt
```

Field Manipulation

bash

```
# Change field separator
awk -F',' '{ print $1, $2 }' file.csv

# Set field separator in script
awk 'BEGIN { FS=":" } { print $1 }' /etc/passwd

# Count fields per line
awk '{ print NF }' file.txt

# Print last field
awk '{ print $NF }' file.txt

# Print all but first field
awk '{ for(i=2; i<=NF; i++) printf "%s ", $i; print "" }' file.txt
```

Conditional Processing

bash

```bash
# Print lines longer than 80 characters
awk 'length > 80' file.txt

# Print lines where first field is numeric and > 100
awk '$1 > 100 && $1 ~ /^[0-9]+$/' file.txt

# Print lines containing specific pattern in field 2
awk '$2 ~ /pattern/' file.txt

# Print lines NOT containing pattern
awk '!/pattern/' file.txt

# Multiple conditions
awk '$1 == "error" && $3 > 100' log.txt
```

Mathematical Operations

```bash
# Sum values in column 1
awk '{ sum += $1 } END { print sum }' file.txt

# Calculate average
awk '{ sum += $1; count++ } END { print sum/count }' file.txt

# Find maximum value
awk '{ if($1 > max) max = $1 } END { print max }' file.txt

# Count occurrences
awk '{ count[$1]++ } END { for(i in count) print i, count[i] }' file.txt
```

String Functions

```bash
```

```
# Convert to uppercase
awk '{ print toupper($0) }' file.txt

# Convert to lowercase
awk '{ print tolower($0) }' file.txt

# String length
awk '{ print length($1) }' file.txt

# Substring
awk '{ print substr($1, 2, 3) }' file.txt

# String substitution
awk '{ gsub(/old/, "new"); print }' file.txt

# Split string
awk '{ split($1, arr, "-"); print arr[1], arr[2] }' file.txt
```

Advanced Examples

Process CSV Files

bash

```
# Print header and specific rows
awk -F',' 'NR==1 || $3 > 1000' data.csv

# Calculate column totals
awk -F',' 'NR>1 { sum += $4 } END { print "Total:", sum }' sales.csv
```

Log File Analysis

bash

```
# Count error types
awk '{ count[$2]++ } END { for(i in count) print i ": " count[i] }' error.log

# Extract IP addresses (assuming they're in field 1)
awk '{ print $1 }' access.log | sort | uniq -c | sort -nr
```

Text Report Generation

bash

```
# Create formatted report
awk 'BEGIN {
    print "Name\t\tScore\tGrade"
    print "----\t\t-----\t-----"
}
{
    if($2 >= 90) grade = "A"
    else if($2 >= 80) grade = "B"
    else if($2 >= 70) grade = "C"
    else grade = "F"
    printf "%-15s %5d\t%s\n", $1, $2, grade
}' students.txt
```

Control Structures

If-Else

bash

```
awk '{
    if($1 > 100)
        print $0 " - High"
    else
        print $0 " - Low"
}' file.txt
```

For Loops

bash

```
# Print all fields with numbering
awk '{ for(i=1; i<=NF; i++) print i ": " $i }' file.txt

# Process array
awk '{
    split($0, arr, " ")
    for(i in arr) print i, arr[i]
}' file.txt
```

While Loops

bash

```
awk '{
  i = 1
  while(i <= NF) {
    print $i
    i++
  }
}' file.txt
```

Useful One-Liners

bash

```
# Remove blank lines
awk 'NF' file.txt

# Print lines between two patterns
awk '/start/,/end/' file.txt

# Print unique lines (like uniq)
awk '!seen[$0]++' file.txt

# Reverse field order
awk '{ for(i=NF; i>=1; i--) printf "%s ", $i; print "" }' file.txt

# Replace multiple spaces with single space
awk '{ $1=$1; print }' file.txt

# Print lines with specific number of fields
awk 'NF == 5' file.txt

# Sum numbers in each line
awk '{ sum=0; for(i=1; i<=NF; i++) sum+=$i; print sum }' file.txt
```

Tips and Best Practices

1. **Use single quotes** to avoid shell interpretation of special characters
2. **Test with small files** before processing large datasets
3. **Use BEGIN/END blocks** for initialization and cleanup
4. **Combine with other tools** using pipes for complex processing
5. **Use -f flag** for complex scripts stored in files

6. **Remember field numbering** starts at 1, not 0
7. **Use regular expressions** for powerful pattern matching
8. **Consider performance** for large files - AWK is generally fast but not always the best choice

Common Pitfalls

- Forgetting that `$0` changes when you modify fields
- Not handling empty lines or missing fields
- Incorrect field separator specification
- Mixing up NR and FNR when processing multiple files
- Not escaping special characters in patterns

See Also

- `sed` - Stream editor for simple text transformations
- `grep` - Pattern matching and searching
- `cut` - Extract specific columns/fields
- `sort` - Sort lines of text
- `uniq` - Remove duplicate lines