Inhoud

Hoofdstuk 4
Zoeken en Analyseren van Tekst (NL)

Bestanden en Directories Zoeken

`find` Commando:

Syntax:
```
find [pad] [criteria] [actie]
```

Voorbeelden van Criteria:

- `-name "bestandsnaam"`: Zoekt naar bestanden met een exacte naam.
- `-iname "bestandsnaam"`: Zoekt case-insensitief.
- `-type f`: Zoekt alleen naar bestanden.
- `-type d`: Zoekt alleen naar directories.
- `-size +100k`: Zoekt naar bestanden groter dan 100 kilobytes.
- `-mtime -7`: Zoekt naar bestanden die gewijzigd zijn in de afgelopen 7 dagen.

Acties:

- `-print`: Standaard actie die het pad van gevonden bestanden afdrukt.
- `-delete`: Verwijdert gevonden bestanden (gebruik voorzichtig).
- `-exec [commando] {} \;`: Voert een commando uit op elk gevonden bestand.

Praktische Toepassing:
```
find /var/log -type f -mtime -1 -exec gzip {} \;
```
Dit commando comprimeert alle logbestanden die in de laatste 24 uur zijn gewijzigd.

`locate` Commando:

Beschrijving:
Zoek snel door gebruik te maken van een vooraf aangemaakte database.

Gebruik:
```
locate [opties] [patroon]
```

Regelmatige Updates:
Gebruik `updatedb` om de database bij te werken.

Opties:

- `-i`: Case-insensitief zoeken.
- `-c`: Tel het aantal gevonden items.

Praktische Toepassing:
`locate -c "*.conf"` geeft het aantal configuratiebestanden in het systeem.

Tekstbestanden Analyseren

`grep` Commando:

Basis Syntax:
```
grep [opties] "patroon" [bestanden]
```

Opties:

- `-i`: Case-insensitief zoeken.
- `-v`: Keert de match om, laat alles zien behalve de patronen.
- `-r`: Recursieve zoekopdrachten door directories.
- `-l`: Toont alleen bestandsnamen met matches.
- `-n`: Toont regelnummers.
- `-E`: Gebruikt uitgebreide reguliere expressies.

Reguliere Expressies:

- `.`: Staat voor elk karakter.
- `^`: Begin van een regel.
- `$`: Einde van een regel.
- `*`: Voorgaande teken 0 of meer keer.
- `[]`: Karakterklasse, bijvoorbeeld `[a-z]`.

Voorbeeld:
`grep -E "^root:" /etc/passwd` vindt regels die beginnen met "root:".

`cut` Commando:

Beschrijving:
Gebruik het `cut`-commando om specifieke kolommen of velden uit een bestand te extraheren.

Syntax:
```
cut -d [scheidingsteken] -f [velden] [bestand]
```

Opties:

- `-d`: Definieert het scheidingsteken, standaard is tab.
- `-f`: Specificeert de velden om te extraheren.
- `--complement`: Neemt alle velden behalve de gespecificeerde.

Praktische Toepassing:
`cut -d ':' -f 1,3 /etc/passwd` haalt gebruikersnamen en hun ID's op.

Tekst Formatteren en Manipuleren

`sort` Commando:

Beschrijving:
Sorteert tekstregels in een bestand.

Syntax:
```
sort [opties] [bestand]
```

Opties:

- `-n`: Sorteert numeriek.
- `-r`: Sorteert omgekeerd.
- `-k`: Sorteert op een specifieke kolom.
- `-t`: Stelt een scheidingsteken in voor kolomsortering.
- `-u`: Verwijdert dubbele regels.

Voorbeeld:
`sort -t ',' -k 3,3 -r data.csv` sorteert een CSV-bestand omgekeerd op de derde kolom.

`uniq` Commando:

Beschrijving:
Verwijdert of rapporteert dubbele regels.

Syntax:
```
uniq [opties] [bestand]
```

Opties:

- `-c`: Telt de voorvallen van elke lijn.
- `-d`: Toont alleen dubbele regels.
- `-u`: Toont alleen unieke regels.

Praktische Toepassing:
`sort namen.txt | uniq -c` toont elke unieke naam en hoe vaak deze voorkomt.

`wc` Commando:

Beschrijving:
Telt regels, woorden en tekens in een bestand.

Syntax:
```
wc [opties] [bestand]
```

Opties:

- `-l`: Telt regels.
- `-w`: Telt woorden.
- `-c`: Telt bytes.

Voorbeeld:
`wc -l verslag.txt` telt het aantal regels in het verslag.

Invoer en Uitvoer Beheren

Omleiden van Uitvoer:

- `>`: Omleidt uitvoer naar een bestand en overschrijft het bestaande bestand.
- `>>`: Voegt uitvoer toe aan het einde van een bestand.
- `2>`: Standaardfout omleiden naar een bestand.

Praktische Toepassing:
`ls /nonexistent > out.txt 2> err.txt` stuurt normale uitvoer naar `out.txt` en fouten naar `err.txt`.

Pipes (|):

Beschrijving:
Verbindt de uitvoer van een commando direct met de invoer van een ander commando.

Praktische Toepassing:
`ps aux | grep "httpd"` zoekt binnen de lijst van actieve processen naar "httpd".

`tee` Commando:

Beschrijving:
Schrijft de uitvoer naar zowel het scherm als naar een bestand.

Syntax:
`tee [opties] [bestand]`

Opties:

- `-a`: Voegt toe aan het bestand in plaats van overschrijven.

Voorbeeld:
`echo "Logregel" | tee -a logboek.txt` toont en voegt een logregel toe.

Bewerken van Tekst

`sed` Commando:

Beschrijving:
Stream editor voor tekstmanipulatie.

Basis Syntax:
```
sed [opties] 'script' [bestand]
```

Opties:

- `-i`: Bewerk het bestand in plaats van uitvoer naar stdout.
- `-e`: Voert een script of commando uit.

Veelgebruikte Scripts:

- `s/patroon/vervanging/g`: Vervangt alle voorvallen van 'patroon' door 'vervanging'.
- `d`: Verwijdert lijnen.

Voorbeeld:
`sed -i 's/foo/bar/g' bestand.txt` vervangt alle voorvallen van "foo" door "bar" in `bestand.txt`.

`awk` Commando:

Beschrijving:
Taal voor tekstverwerking en gegevensmanipulatie.

Basis Syntax:
```
awk 'script' [bestand]
```

Praktische Toepassingen:

- **Kolomselectie:**
  `awk -F ':' '{print $1, $3}' /etc/passwd` haalt gebruikersnamen en hun ID's op.
- **Gegevensverwerking:**
  `awk '{s+=$1} END {print s}' cijfers.txt` berekent de som van de eerste kolom.
- **Conditioneel Uitvoeren:**
  `awk '$3 > 50 {print $1}' cijfers.txt` print alleen de eerste kolom als de derde kolom groter is dan 50.

Geavanceerde Gebruikstoepassingen

Reguliere Expressies:

Basisconcepten:

- `.`: Matcht elk enkel karakter.
- `*`: Matcht 0 of meer van het voorgaande element.
- `[]`: Matcht een reeks van karakters.
- `^`: Begin van een regel.
- `$`: Einde van een regel.

Praktische Toepassing:
`grep -E "^[A-Za-z]+" namen.txt` vindt lijnen die beginnen met een letter.

Shellscripts en Automatisering:

Beschrijving:
Gebruik shellscripts om taken te automatiseren, herhaalbare processen te stroomlijnen en consistentie te waarborgen.

Structuur:

- Begin met `#!/bin/bash` om de shell aan te geven.
- Gebruik `echo` voor uitvoer en `read` voor invoer.
- Maak gebruik van conditionele statements (if, then, else) en loops (for, while) voor logica.

Praktische Toepassing:
Schrijf een shellscript om bijvoorbeeld een bestand te analyseren en actie te ondernemen op basis van de inhoud, zoals logbestanden controleren op fouten en een melding sturen als er een fout wordt gevonden.

```
for file in *.log; do
        echo "Verwerken van $file"
        grep "ERROR" $file | tee -a errors.txt
done
```

Chapter 4 Searching and Analyzing tekst (EN)

Searching Files and Directories

`find` Command:

Syntax:
```
find [path] [criteria] [action]
```

Examples of Criteria:

- `-name "filename"`: Searches for files with an exact name.
- `-iname "filename"`: Searches case-insensitively.
- `-type f`: Searches only for files.
- `-type d`: Searches only for directories.
- `-size +100k`: Searches for files larger than 100 kilobytes.
- `-mtime -7`: Searches for files modified in the past 7 days.

Actions:

- `-print`: The default action that prints the path of the found files.
- `-delete`: Deletes the found files (use with caution).
- `-exec [command] {} \;`: Executes a command on each found file.

Practical Application:
```
find /var/log -type f -mtime -1 -exec gzip {} \;
```
This command compresses all log files that have been modified in the last 24 hours.

`locate` Command:

Description:
Searches quickly using a pre-built database.

Usage:
```
locate [options] [pattern]
```

Regular Updates:
Use `updatedb` to update the database.

Options:

- `-i`: Case-insensitive search.
- `-c`: Counts the number of matching items.

Practical Application:
`locate -c "*.conf"` counts the number of configuration files in the system.

Analyzing Text Files

`grep` Command:

Basic Syntax:
```
grep [options] "pattern" [files]
```

Options:

- `-i`: Case-insensitive search.
- `-v`: Inverts the match, showing everything except the patterns.
- `-r`: Recursive searches through directories.
- `-l`: Shows only filenames with matches.
- `-n`: Displays line numbers.
- `-E`: Uses extended regular expressions.

Regular Expressions:

- `.`: Matches any single character.
- `^`: Start of a line.
- `$`: End of a line.
- `*`: Matches the preceding character 0 or more times.
- `[]`: Character class, e.g., `[a-z]`.

Example:
`grep -E "^root:" /etc/passwd` finds lines that start with "root:".

`cut` Command:

Description:
Extracts specific columns or fields from a file.

Syntax:
```
cut -d [delimiter] -f [fields] [file]
```

Options:

- `-d`: Defines the delimiter, default is tab.
- `-f`: Specifies the fields to extract.
- `--complement`: Takes all fields except the specified ones.

Practical Application:
`cut -d ':' -f 1,3 /etc/passwd` retrieves usernames and their IDs.

Formatting and Manipulating Text

`sort` Command:

Description:
Sorts text lines in a file.

Syntax:
```
sort [options] [file]
```

Options:

- `-n`: Sorts numerically.
- `-r`: Sorts in reverse order.
- `-k`: Sorts by a specific column.
- `-t`: Sets a delimiter for column sorting.
- `-u`: Removes duplicate lines.

Example:
`sort -t ',' -k 3,3 -r data.csv` sorts a CSV file in reverse order by the third column.

`uniq` Command:

Description:
Removes or reports duplicate lines.

Syntax:
```
uniq [options] [file]
```

Options:

- `-c`: Counts occurrences of each line.
- `-d`: Shows only duplicate lines.
- `-u`: Shows only unique lines.

Practical Application:
`sort names.txt | uniq -c` shows each unique name and how often it occurs.

`wc` Command:

Description:
Counts lines, words, and characters in a file.

Syntax:
```
wc [options] [file]
```

Options:

- `-l`: Counts lines.
- `-w`: Counts words.
- `-c`: Counts bytes.

Example:
`wc -l report.txt` counts the number of lines in the report.

Managing Input and Output

Output Redirection:

- `>`: Redirects output to a file, overwriting the existing file.
- `>>`: Appends output to the end of a file.
- `2>`: Redirects standard error to a file.

Practical Application:
`ls /nonexistent > out.txt 2> err.txt` sends normal output to `out.txt` and errors to `err.txt`.

Pipes (|):

Description:
Connects the output of one command directly to the input of another command.

Practical Application:
`ps aux | grep "httpd"` searches within the list of active processes for "httpd".

`tee` Command:

Description:
Writes output to both the screen and a file.

Syntax:
`tee [options] [file]`

Options:

- `-a`: Appends to the file instead of overwriting.

Example:
`echo "Log entry" | tee -a logbook.txt` shows and appends a log entry.

Editing Text

`sed` Command:

Description:
Stream editor for text manipulation.

Basic Syntax:
`sed [options] 'script' [file]`

Options:

- `-i`: Edits the file in-place instead of outputting to stdout.
- `-e`: Executes a script or command.

Common Scripts:

- `s/pattern/replacement/g`: Replaces all occurrences of 'pattern' with 'replacement'.
- `d`: Deletes lines.

Example:
`sed -i 's/foo/bar/g' file.txt` replaces all occurrences of "foo" with "bar" in `file.txt`.

`awk` Command:

Description:
A language for text processing and data manipulation.

Basic Syntax:
`awk 'script' [file]`

Practical Applications:

- **Column Selection:**
  `awk -F ':' '{print $1, $3}' /etc/passwd` retrieves usernames and their IDs.
- **Data Processing:**
  `awk '{s+=$1} END {print s}' numbers.txt` calculates the sum of the first column.
- **Conditional Execution:**
  `awk '$3 > 50 {print $1}' numbers.txt` prints the first column only if the third column is greater than 50.

Advanced Use Cases

Regular Expressions:

Basic Concepts:

- `.`: Matches any single character.
- `*`: Matches 0 or more of the preceding element.
- `[]`: Matches a range of characters.
- `^`: Start of a line.
- `$`: End of a line.

Practical Application:

`grep -E "^[A-Za-z]+" names.txt` finds lines that start with a letter.

Shell Scripts and Automation:

Description:

Create scripts to automate tasks, streamline repeatable processes, and ensure consistency.

Structure:

- Start with `#!/bin/bash` to indicate the shell.
- Use `echo` for output and `read` for input.
- Use conditional statements (`if`, `then`, `else`) and loops (`for`, `while`) for logic.

Practical Application:

Write a shell script to analyze a file and take action based on its contents, such as checking log files for errors and sending a notification if an error is found.

```
for file in *.log; do
        echo "Verwerken van $file"
        grep "ERROR" $file | tee -a errors.txt
done
```