### **Linux Recap**

### **File Viewing and Editing**

#### **Viewing Files**

1. **cat (concatenate)**
   * **Usage**: Displays the contents of a file to the terminal.

**Example**:  
  
cat filename.txt

* + **Use Case**: Quickly viewing the entire content of small files.

1. **more**
   * **Usage**: Similar to cat, but allows scrolling through the content page by page.

**Example**:  
  
more filename.txt

* + **Use Case**: Viewing large files where you need to control the display of content.

1. **less**
   * **Usage**: Advanced version of more with both forward and backward navigation.

**Example**:  
  
less filename.txt

* + **Use Case**: Better for large files, as it does not load the entire file into memory.

1. **head**
   * **Usage**: Displays the first few lines of a file.

**Example**:  
  
head filename.txt

head -n 20 filename.txt # Display first 20 lines

* + **Use Case**: Quickly checking the beginning of log files or configuration files.

1. **tail**
   * **Usage**: Displays the last few lines of a file.

**Example**:  
  
tail filename.txt

tail -n 20 filename.txt # Display last 20 lines

* + **Use Case**: Monitoring the end of log files for recent entries, especially in live updates with tail -f filename.txt.

#### **Editing Files**

1. **nano**
   * **Usage**: Simple and user-friendly text editor.

**Example**:  
  
nano filename.txt

* + **Use Case**: Suitable for quick edits with intuitive shortcuts.

1. **vi (or vim)**
   * **Usage**: Powerful text editor with two modes: command and insert.

**Example**:  
  
vi filename.txt

* + - Press i to enter insert mode.
    - Press Esc to return to command mode.
    - Type :wq to save and exit, or :q! to exit without saving.
  + **Use Case**: Ideal for advanced users needing powerful text manipulation capabilities.

### **Basic Text Processing**

#### **Searching with grep**

* **Usage**: Searches for patterns within files.

**Example**:  
  
grep 'search\_term' filename.txt

grep -i 'search\_term' filename.txt

grep -r 'search\_term' /path/to/directory

* **Use Case**: Finding specific information in logs, configuration files, or any text files.

#### **Basic Text Manipulation**

1. **cut**
   * **Usage**: Cuts out sections from each line of files.

**Example**:  
  
cut -d ',' -f 1,3 filename.csv # Cut columns 1 and 3 from a CSV

* + **Use Case**: Extracting specific columns from data files.

1. **sort**
   * **Usage**: Sorts lines of text files.

**Example**:  
  
sort filename.txt

sort -r filename.txt # Reverse sort

sort -n filename.txt # Numeric sort

* + **Use Case**: Organizing data in files by alphabetical, numerical, or custom order.

1. **uniq**
   * **Usage**: Reports or filters out repeated lines in a file.

**Example**:  
  
sort filename.txt | uniq # Remove duplicate lines

sort filename.txt | uniq -c # Count occurrences of each line

* + **Use Case**: Cleaning up data by removing duplicates.

### **Examples and Use Cases**

1. **Viewing Logs for Errors**

Use grep to search for error patterns in logs:  
  
grep 'ERROR' /var/log/syslog

Use tail -f to monitor real-time log updates:  
  
tail -f /var/log/syslog

1. **Analyzing CSV Data**

Extract specific columns with cut:  
  
cut -d ',' -f 1,2 data.csv

Sort data and remove duplicates:  
  
sort data.csv | uniq

1. **Quick Edits**

Open and edit a configuration file with nano:  
  
nano /etc/apache2/apache2.conf

1. **Advanced Text Editing**

Edit a script with vi:  
  
vi script.sh

* + - Add or modify lines, save changes, and exit.

1. **Combining Commands for Powerful Text Processing**

Extract, sort, and count unique IP addresses from a log file:  
  
cut -d ' ' -f 1 access.log | sort | uniq -c | sort -nr

* + Explanation:
    - cut -d ' ' -f 1 access.log: Extract the first column (assumed to be IP addresses).
    - sort: Sort the IP addresses.
    - uniq -c: Count occurrences of each IP.
    - sort -nr: Sort the counts in numeric, reverse order to see the most frequent IPs first.

1. **Creating a Shell Script**
   * Use a text editor to create a new file with a .sh extension.

Example:  
  
nano myscript.sh

Add commands to the script:  
  
#!/bin/bash

echo "Hello, World!"

1. **Running a Shell Script**

Make the script executable:  
  
chmod +x myscript.sh

Execute the script:  
  
./myscript.sh

Output:  
  
Hello, World!

### **Understanding Shebang (#!)**

* **Shebang (#!)**
  + The first line of a shell script typically starts with #!/path/to/shell.
  + It tells the system which interpreter to use to execute the script.

Common shebang for Bash:  
  
#!/bin/bash

## **Shell Scripting Fundamentals**

### **Variables**

#### **Defining Variables**

* **Syntax**: VARIABLE\_NAME=value

Example:  
  
NAME="John"

AGE=25

#### **Using Variables**

* **Accessing Variables**: Use the $ symbol before the variable name.

Example:  
  
echo "My name is $NAME and I am $AGE years old."

Output:  
  
My name is John and I am 25 years old.

#### **Environment Variables**

* **Definition**: Environment variables are global variables available to any child process of the shell.

**Setting Environment Variables**:  
  
export VARIABLE\_NAME=value

Example:  
  
export PATH=$PATH:/new/path

### **Basic I/O Operations**

#### **echo and read Commands**

* **echo**: Used to display text or variables.

Example:  
  
echo "Hello, World!"

echo "My name is $NAME."

* **read**: Used to take input from the user.

Example:  
  
echo "Enter your name: "

read USER\_NAME

echo "Hello, $USER\_NAME!"

#### **Redirecting Output and Input**

* **Output Redirection**
  + **Syntax**: command > file (overwrite) or command >> file (append)

Example:  
  
echo "Hello, World!" > output.txt

echo "This is a new line." >> output.txt

* **Input Redirection**
  + **Syntax**: command < file

Example:  
  
while read line

do

echo $line

done < input.txt

### **Example Shell Script**

Here is a simple shell script demonstrating variables, I/O operations, and redirection:

#!/bin/bash

# Define variables

NAME="Alice"

AGE=30

# Output variables

echo "Name: $NAME"

echo "Age: $AGE"

# Prompt user for input

echo "Enter your favorite color: "

read COLOR

echo "Your favorite color is $COLOR"

# Redirect output to a file

echo "User details:" > user\_details.txt

echo "Name: $NAME" >> user\_details.txt

echo "Age: $AGE" >> user\_details.txt

echo "Favorite color: $COLOR" >> user\_details.txt

# Display the content of the file

cat user\_details.txt

## **Conditional Statements**

### **if, else, elif Constructs**

1. **if**

**Syntax**:  
  
if [ condition ]; then

# commands

fi

**Example**:  
  
if [ $AGE -ge 18 ]; then

echo "You are an adult."

fi

1. **if-else**

**Syntax**:  
  
if [ condition ]; then

# commands

else

# commands

fi

**Example**:  
  
if [ $AGE -ge 18 ]; then

echo "You are an adult."

else

echo "You are a minor."

fi

1. **if-elif-else**

**Example**:  
  
if [ $AGE -lt 13 ]; then

echo "You are a child."

elif [ $AGE -ge 13 ] && [ $AGE -lt 18 ]; then

echo "You are a teenager."

else

echo "You are an adult."

fi

### **Using Test Conditions**

**Example**:  
  
if [ $AGE -ge 18 ]; then

echo "You are an adult."

fi

### **Example of Conditions**

#!/bin/bash

echo "Enter your age: "

read AGE

if [[ $AGE -lt 13 ]]; then

echo "You are a child."

elif [[ $AGE -ge 13 && $AGE -lt 18 ]]; then

echo "You are a teenager."

else

echo "You are an adult."

fi

## **Loops**

### **for, while, and until Loops**

1. **for Loop**

**Example**:  
  
for i in 1 2 3 4 5; do

echo "Iteration $i"

done

1. **while Loop**

**Example**:  
  
COUNTER=0

while [ $COUNTER -lt 5 ]; do

echo "Counter is $COUNTER"

COUNTER=$((COUNTER + 1))

done

1. **until Loop**

**Syntax**:  
  
until [ condition ]; do

# commands

done

**Example**:  
  
COUNTER=0

until [ $COUNTER -ge 5 ]; do

echo "Counter is $COUNTER"

COUNTER=$((COUNTER + 1))

done

### **Loop Control: break, continue**

1. **break**
   * **Usage**: Exits the loop immediately.

**Example**:  
  
for i in 1 2 3 4 5; do

if [ $i -eq 3 ]; then

break

fi

echo "Iteration $i"

done

# Output: Iteration 1 Iteration 2

1. **continue**
   * **Usage**: Skips the rest of the commands in the current loop iteration and moves to the next iteration.

**Example**:  
  
for i in 1 2 3 4 5; do

if [ $i -eq 3 ]; then

continue

fi

echo "Iteration $i"

done

# Output: Iteration 1 Iteration 2 Iteration 4 Iteration 5

### **Example Script with Loops and Conditionals**

#!/bin/bash

# Prompt the user for a number

echo "Enter a number between 1 and 10: "

read NUMBER

# Check if the number is within the range

if [[ $NUMBER -lt 1 || $NUMBER -gt 10 ]]; then

echo "Number is out of range."

exit 1

fi

# for loop

for i in {1..10}; do

echo "for loop iteration $i"

done

# while loop

COUNTER=1

while [[ $COUNTER -le 10 ]]; do

echo "while loop iteration $COUNTER"

COUNTER=$((COUNTER + 1))

done

# until loop

COUNTER=1

until [[ $COUNTER -gt 10 ]]; do

echo "until loop iteration $COUNTER"

COUNTER=$((COUNTER + 1))

done

# Using break and continue

for i in {1..10}; do

if [[ $i -eq 5 ]]; then

echo "Breaking the loop at iteration $i"

break

fi

if [[ $i -eq 3 ]]; then

echo "Skipping iteration $i"

continue

fi

echo "Iteration $i"

done