Bryant F. Polanco

Question 1

awk

Description:

Awk is a versatile programming language used for pattern scanning and processing.

Formula/Syntax:

```
awk 'pattern { action }' file
```

Examples:

```
awk '{ print $1 }' data.txt     # Print the first column of a file
awk '/pattern/' file.txt        # Print lines matching a pattern
awk '{ sum += $1 } END { print sum }' numbers.txt   # Calculate and print
the sum of the first column
```

cat

Description:

Concatenate and display the content of files.

Formula/Syntax:

```
cat file1 file2
```

Examples:

```
cat file.txt                    # Display the content of a file
cat file1.txt file2.txt > newfile.txt   # Concatenate two files into a new
file
cat *.txt > combined.txt         # Concatenate all .txt files into a single
file
```

cp

Description:

Copy files or directories.

Formula/Syntax:

```
cp source destination
```

Examples:

```
cp file.txt backup/             # Copy a file to a directory
cp -r dir1 dir2                 # Copy a directory and its contents
recursively
cp *.txt backup/                # Copy all .txt files to a backup directory
```

cut

Description:

Remove sections from each line of a file.

Formula/Syntax:

```
cut -d delimiter -f fields file
```

Examples:

```
cut -d',' -f1,3 data.csv        # Extract first and third columns from a
CSV file
cut -c 1-5 file.txt             # Extract the first five characters from
each line
cut -f 1-3,5 data.tsv           # Extract specified fields from a tab-
delimited file
```

grep

Description:

Search for patterns in files.

Formula/Syntax:

```
grep pattern file
```

Examples:

```
grep "error" logfile.txt        # Search for lines containing the word
"error"
grep -i "pattern" file.txt       # Case-insensitive search for a pattern
grep -r "search" /path/to/dir    # Recursively search for a pattern in
files within a directory
```

head

Description:

Display the first lines of a file.

Formula/Syntax:

```
head -n N file
```

Examples:

```
head -n 10 data.txt             # Display the first 10 lines of a file
head -n 20 *.log                # Display the first 20 lines of all log
files in a directory
head -n 30 data.txt             # Display the first 30 lines of a file
```

ls

Description:

List directory contents.

Formula/Syntax:

```
ls options directory
```

Examples:

```

```

```
ls -l                           # Long format listing
ls -a                           # List all files, including hidden ones
ls *.txt                        # List all .txt files in the current
directory
```

man

Description:

Display the manual for a command.

Formula/Syntax:

```
man command
```

Examples:

```
man ls                          # Display the manual for the "ls" command
man grep                        # Display the manual for the "grep" command
man tr                          # Display the manual for the "tr" command
```

mkdir

Description:

Create directories.

Formula/Syntax:

```
mkdir directory
```

Examples:

```
mkdir new_folder                # Create a new directory named "new_folder"
mkdir -p path/to/nested/dir     # Create nested directories with -p option
mkdir Downloads/Games           # Create a new directory named "Games"
```

mv

Description:

Move or rename files and directories.

Formula/Syntax:

```
mv source destination
```

Examples:

```
mv file.txt new_location/      # Move a file to a different directory
mv old_name.txt new_name.txt   # Rename a file
mv ~/Documents/sold_name.txt ~/Documents/new_name.txt   # Rename a file
using absolute path
```

tac

Description:

Concatenate and display the content of files in reverse.

Formula/Syntax:

```
tac file
```

Examples:

```
tac file.txt                  # Display the content of a file in reverse
order
tac *.log > reversed_logs.txt   # Concatenate and reverse the content of
all log files
```

tail

Description:

Display the last lines of a file.

Formula/Syntax:

bash

```
tail -n N file
```

Examples:

```
tail -n 15 data.txt            # Display the last 15 lines of a file
tail -f log_file.txt           # Display and follow the content of a log
file
tail -n 20 data.txt            # Display the last 20 lines of a file
```

touch

Description:

Create an empty file or update file timestamps.

Formula/Syntax:

```
touch filename
```

Examples:

```
touch new_file.txt             # Create a new empty file
touch -c existing_file.txt     # Update the timestamp of an existing file
touch "file with space"        # Create a file with a space in its name
```

tr

Description:

Translate or delete characters.

Formula/Syntax:

```
tr options set1 set2
```

Examples:

```
echo "Hello" | tr 'a-z' 'A-Z'  # Convert lowercase to uppercase
echo "12345" | tr -d '2'       # Delete the character '2' from the input
cat file.txt| tr '.' ','       # Translates one character to another
```

tree

Description:

Display directory tree structure.

Formula/Syntax:

```
tree options directory
```

Examples:

```
tree                              # Display the directory tree structure of
the current directory
tree -L 2 /path/to/dir            # Display the tree structure
tree -a ./GFG                     # Display the tree hierarchy of a directory
(Taken from geeksforgeeks.com)
```

Question 2

Opening a New Terminal Tab:

```
GNOME Terminal: Ctrl + Shift + T
iTerm (macOS): Cmd + T
```

To access manual pages, you can use the man command:

```
man command                     # Replace "command" with the command you want
to learn about.
```

How to Parse (Search) for Specific Words in the Manual Page

You can use the man command along with grep to search for specific words:

```
man command | grep keyword     # Replace "command" with the command's manual
you are interested in, and "keyword" with the word you want to search.
```

How to Redirect Output (> and |)

Redirecting Output to a File (>):

```
command > output.txt          # This will redirect the output of "command"
to a file named "output.txt."
```

Piping Output to Another Command (|):

```
command1 | command2           # This pipes the output of "command1" as
input to "command2."
```

How to append Output of a Command to a File

To append output to a file, use the >> operator:

```
command >> output.txt         # This will append the output of "command" to
the end of the file "output.txt."
```

How to Use Wildcards

Wildcards allow you to match filenames with patterns. Common wildcards include:

*: Matches any sequence of characters. ?: Matches any single character. [...]: Matches any single character within the brackets.

Examples:

```
ls *.txt          # List all files with a .txt extension
rm file?.txt      # Remove files like file1.txt, file2.txt
```

Copying multiple files at the same time

```
cp *.txt ~/Downloads         # This copies all .txt files
```

Moving Multiple Files:

```
mv *.txt ~/Downloads/textfiles         # This moves all .txt files to
textfiles
```

Using Brace Expansion

Brace expansion allows you to generate multiple strings with similar patterns:

```
echo {1..5}                 # Outputs: 1 2 3 4 5
```

Creates files: file1.txt, file2.txt, file3.txt

Creating Entire Directory Structures in a Single Command

Use the mkdir -p command to create parent directories as needed:

```
mkdir -p Downloads/PirateGames/{EA, Steam}       # This creates the
specified directory and any missing parent directories along the path.
```