Pipes: Building Powerful Workflows in Bash

Pipes are one of the most powerful features of the Bash shell. They allow you to chain commands together, sending the output of one command as the input for the next. This lets you perform complex data processing in a single line or short script.

Use Cases:

1. Filtering Files:

```
ls | grep .txt
```

This classic example lists all files ending with ".txt" in the current directory. `ls` provides a list, piped (`|`) to `grep` which searches for lines containing ".txt".

```
ps aux | grep keyword
```

This finds processes containing a specific keyword in their name. `ps aux` lists all processes, piped to `grep` to filter for those with the keyword.

2. Analyzing Data:

```
cat log.txt | grep error | wc -l
```

This counts the number of lines containing "error" in the file "log.txt". `cat` reads the file, piped to `grep` to filter for errors, then piped to `wc -l` for line count.

`df -h | sort -n`

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# df -h | sort -n
/dev/sda1        79G   34G   41G  46% /
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           3.9G     0  3.9G   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           795M  128K  795M   1% /run/user/1000
tmpfs           795M  1.3M  793M   1% /run
udev            3.9G     0  3.9G   0% /dev
```

This sorts disk usage information by used space in human-readable format. `df -h` shows disk usage, piped to `sort -n` to sort numerically by the second column (used space).

3. Chained Processing:

`cat oui36.csv | cut -d ',' -f 2 | sort -u`

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat oui36.csv | cut -d ',' -f 2 | sort -u
001BC5000
001BC5001
001BC5002
001BC5003
001BC5004
001BC5005
001BC5006
001BC5007
001BC5008
001BC5009
001BC500A
001BC500B
001BC500C
001BC500D
001BC500E
001BC500F
001BC5010
001BC5011
001BC5012
001BC5013
001BC5014
001BC5015
001BC5016
```

This extracts the second field (comma-separated) from "data.csv" and sorts unique entries. `cat` reads the file, piped to `cut` to extract the second field, then piped to `sort -u` for unique entries.

4. Combining with Other Techniques:

```
find . -type f | grep .log | xargs grep error
```

This finds all ".log" files recursively and searches for lines with "error". `find` searches for files, piped to `grep` to filter logs, then piped to `xargs grep` for another round of filtering within the logs.

Tips for Mastering Pipes:

- **Readability:** Break down complex pipelines into smaller, more manageable steps. Use temporary files if needed for intermediate results.
- **Error Handling:** Consider using options like `grep -i` (case-insensitive) or `||` (OR operator) for handling potential errors.
- **Practice**: Experiment with different commands and pipe combinations to solve real-world tasks.