Notes 9

File Manipulation Continued

grep

Definition

Globally match a **r**egular **e**xpression, and **p**rints the output.

Usage/Formula

Use to search a file or command output for a string.

Examples

```
grep "admin" access.log
```

> Finds every instance of the regular expression "admin" in the access log file.

```
grep -vc "SUCCESS" access.log
```

> `v` shows all non-matching to expression "SUCCESS", -c prints a count

```
grep -i "judy" users.csv
```

> Searches with case insensitivity

```
grep -nr ./
```

> `-r` searches recursively through the current directory through all files, with line number `-n`

awk

Definition

Programming language meant for complex text processing.

Usage/Formula

Used for pattern matching and data manipulation, as opposed to cut which is limited to simple text manipulation.

Uses whitespace by default for separation between columns, can specify field delimiters using the `-F` option.

Examples

```
awk '{print $3}' chatlogs.txt
```

> prints the third field of the `chatlogs.txt`

```
awk -F',' '{print $1, $5, $6}' materials.csv
```

> prints first, fifth, and sixth field, separated by `,` with the `-F` flag

```
awk '{sum += $3; count++} END {print "Salary average:", sum/count}'
employees.csv
```

> Totals the third column, counts the amount of rows, then prints `Salary average:` followed by the sum divided by rowcount

sed

Definition

Stream editor, used to do basic text transforms on an input stream such as a file or from a piped input.

Usage/Formula

Adding characters, changing a certain specified string.

Examples

```
sed 's/192.168.10.204/& SUSPICIOUS/g' access.log
```

> Flag IPs without duplicating pattern using &

```
sed 's/^port=.*/port=3000/' nginx.conf
```

> Use regex to keep front portion of search and replace
> `.` starts the area to be replaced
> as many characters to end of line `*`
> that portion replaced by `3000`

```
sed -i.bak 's/^DB_HOST=localhost$/DB_HOST=10.0.40.8/g' .env
```

> creates backup with the in place edit `-i`, with `.bak` suffix applied

Pipes

Usage

Transfer string from one command to the next. Allows for multiple commands to run after each other in sequence, making single commands stronger together.

Examples

```
man grep | grep "ignore"
```

> pipes output of `man grep` to `grep` searching for all instances of ignore

```
ls -l | head -5
```

> shows files in a long format, then shows only the first five lines of `ls`

```
grep -i "maersk" ship-manifest2025-07-14.log | awk '{print $5}' | sort -n |
uniq -c | sort -rn
```

> finds `"maersk"` in the file, prints the fifth field, sorts by number, counts unique, then sorts by frequency (highest first)

Saving output of command

Usage

Saves the output of a command to a file, using the `>` character.

Examples

```
tree -L2 > dir_structure.txt
```

> Saves the output of `tree` to `dir_structure.txt`

```
ip addr show | grep -e "192.168" -e "10.0" | awk '{print $2}' >
private_ip_addr.txt
```

> Prints the ip addresses from searching in `ip addr show` cmd for private ip subnets.

```
cut -d',' -f2,6 games.csv | sort > games_and_version.txt
```

> Slices `games.csv` by field 2 and 6, sorts, then saves to `games_and_version.txt`

Appending output of command

Usage

Appends command to the end of a file using `>>`.

Examples

```
echo "$(date --rfc-3339=seconds) starting update" >> update.log
sudo dnf update
echo "$(date --rfc-3339=seconds) update complete" >> update.log
```

> appends to update.log for tracking update timestamps

```
grep "ERROR" nginx.log >> all_errors.log
grep "ERROR" minecraft_server.log >> all_errors.log
grep "ERROR" jellyfin_headless.log >> all_errors.log
```

> Appends all errors from logs to `all_errors.log`

```
# env essentials
echo "System scan done: $(date --rfc-3339=seconds)" >> env_hw.txt
env | grep -e "SHELL" -e "NAME" >> env_hw.txt
lscpu | grep -i "model name" >> env_hw.txt
lspci | grep -i "ethernet" >> env_hw.txt
```

> Small script that appends shell directory, cpu model, and ethernet controller to a script that lists hardware.

awk csv extraction

```
awk -F',' -k '$9 == "APPROVED" {print $5,$6,$9}' new_users.csv >>
whitelist.csv
```

> Logically prints lines if `APPROVED` in field `$9` in `new_users.csv` file
> Appends to whitelist.csv