

# Notes 9

## Handling text files part 2

### The GREP command

The **grep** command is used to search text in a given file. Grep works by line basis (it matches the search criteria in a line by line basis).

- Usage:

- `grep + option + search criteria + file(s)`

- Basic Example:

- Search any line that contains the word 'Game' in the given file:

- `grep 'Game' ~/Documents/GameofThrones.txt`

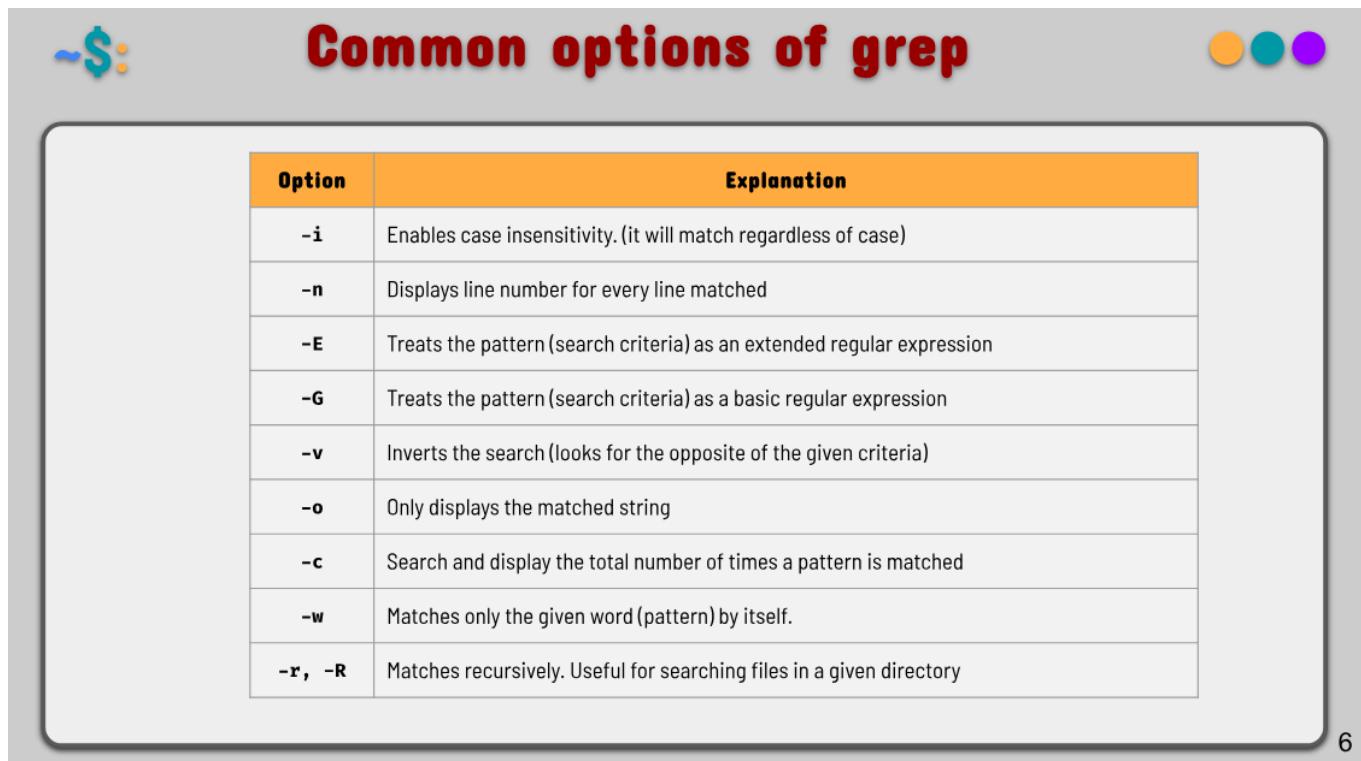
- Search any line that contains the word 'thrones' regardless of the case.

- `grep -i 'Thrones' ~/Documents/Books/GameofThrones.txt`

- Display how many lines contain the matched string:

- `grep -c 'Game' ~/Documents/Books/GameofThrones.txt`

- More options for **grep**:



**Common options of grep**

Option	Explanation
<code>-i</code>	Enables case insensitivity. (it will match regardless of case)
<code>-n</code>	Displays line number for every line matched
<code>-E</code>	Treats the pattern (search criteria) as an extended regular expression
<code>-G</code>	Treats the pattern (search criteria) as a basic regular expression
<code>-v</code>	Inverts the search (looks for the opposite of the given criteria)
<code>-o</code>	Only displays the matched string
<code>-c</code>	Search and display the total number of times a pattern is matched
<code>-w</code>	Matches only the given word (pattern) by itself.
<code>-r, -R</code>	Matches recursively. Useful for searching files in a given directory

- More examples of **grep**

- Search any line that contains the word dracula regardless of case and with number line:

- `grep -in 'dracula' ~/Documents/Books/dracula.txt`

- Search for all the lines that do not contain the word 'war':

- `grep -v 'war' ~/Documents/Books/war-and-peace.txt`
- Search and display only matched string (pattern)
  - `grep -o 'pride' ~/Documents/Books/Pride&Prejudice'`
- Display a list of users with the /bin/bash login shell:
  - `grep -i "/bin/bash" /etc/passwd`
- Display our user's information as stored in the /etc/passwd
  - `grep -i $USER /etc/passwd`
- Expanding on `grep`



## Even more examples of grep



- Search for a given strings inside files in a given directory.
  - `grep -iR 'conf' /etc/`
- Search and display the total number of times a given word appears in a file
  - `grep -wc '/bin/bash' /etc/passwd`
- The ^ (caret) symbol matches the empty string at the beginning of a line. Search for all the lines that start with a given word.
  - `grep -ni '^dracula' ~/Documents/Books/dracula.txt`
- Search for all the lines that ends with the string "nologin"
  - `grep -n 'nologin$' /etc/passwd`

8



## Is there no end to grep?



- Search for all the lines that start with a capital letter
  - `grep -n '^[A-Z]' ~/Documents/Books/war-and-peace.txt`
- Search for more than one word per line
  - `grep -Ew 'horror|love|scare' ~/Documents/Books/dracula.txt`
- Match only lines containing IPv4 addresses
  - `grep -E '^[[[:digit:]]{1,3}\.[[[:digit:]]{1,3}\.[[[:digit:]]{1,3}\.[[[:digit:]]{1,3}}' Documentation.txt`
- Search all lines that contain a character repeated 3 times
  - `grep -E "A{3}" file.txt`
- Search all lines that contain a phone number of the format 973-111-2222
  - `grep "[[:digit:]]\{3\}[-][[:digit:]]\{3\}[-][[:digit:]]\{4\}" contacts.csv`
- Display only the options of the of any command from its man page
  - `man ls | grep "^[[:space:]]*[[:punct:]]"`

9



## And grep continues



- Search for all the lines that contain a single word in all the files in your system
  - `grep -niR '^linux$' /`
- The .(period) symbol is a meta-character that matches any single character. Search for all the lines that contain a word that starts with letter d has 4 characters after.
  - `grep -n '^d....' ~/Documents/Books/dracula.txt`
- Bracket expressions allows match a group of characters by enclosing them in brackets [ ]. Match all the lines that contain a the words list, last, lost.
  - `grep -n 'l[aio]st' ~/Documents/Books/dracula.txt`

1  
0

## The AWK command

The **awk** command is a scripting language used for processing and displaying text. Awk can work with a text file from standard output. There are several implementations of Awk, like: nawk, mawk, gawk, and busybox. It performs operations line by line.

- Usage:
  - `awk + options + {awk command} + file + file to save (optional)`
- Basic example:
  - Print the first column of every line of a file:
    - `awk '{print $1}' ~/Documents/Csv/cars.csv'`
- Awk Variables:



<b>\$0</b>	Whole line	
<b>\$1, \$2..\$NF</b>	First, second... last field	
<b>NR</b>	Total Number of Records	
<b>NF</b>	N number of Fields	
<b>OFS</b>	Output Field Separator (default " ")	
<b>FS</b>	input Field Separator (default " ")	
<b>ORS</b>	Output Record Separator (default "\n")	
<b>RS</b>	input Record Separator (default "\n")	
<b>FILENAME</b>	Name of the file	
<b>ARGC</b>	Number of arguments	
		<b>ARGV</b> Array of arguments
		<b>FNR</b> File Number of Records
		<b>OFMT</b> Format for numbers (default "%.6g")
		<b>RSTART</b> Location in the string
		<b>RLENGTH</b> Length of match
		<b>SUBSEP</b> Multi-dimensional array separator (default "\034")
		<b>ARGIND</b> Argument Index
		<b>ENVIRON</b> Environment variables
		<b>IGNORECASE</b> Ignore case
		<b>CONVFMT</b> Conversion format
		<b>ERRNO</b> System errors
		<b>FIELDWIDTHS</b> Fixed width fields

1  
4

- Examples on awk:

- Print first field of /etc/passwd file
  - awk -F: '{print \$1}' /etc/passwd
- Print the last field of the /etc/passwd file
  - awk -F: '{print \$NF}' /etc/passwd
- Print the first and last field of the /etc/passwd
  - awk -F: '{print \$1, " = ", \$NF}' /etc/passwd
- Print the first 3 and 3 field with line numbers
  - awl -F: '{print NR,\$1,\$3}' /etc/passwd
- Print the first and 4th field with a different field separator
  - awk -F: '{OFS="="}{print \$1,\$4}' /etc/passwd
- Start printing a file from a given line (exclude the first 2 lines)
  - awk 'NR > 3 { print }' /etc/passwd

- More examples of awk:

# ~\$: Even more examples of awk

- Interesting example of awk:

Applications Dec 4 5:30 PM 29% 75% 12% 4.9 kB 1.4 kB

Week Report 9 | Linux Funda Handling Text Files Part 2 Google Drive - Denunciar Institution Page

docs.google.com/presentation/d/e/2PACX-1vQnsW6oddLgq6wJLRu0E\_g\_cuDEudz1RLnD1UA

# ~\$: A more interesting example of awk

```
+ ls -lhd --time-style=+MD /* | awk -v OFS="\t" 'BEGIN { print "Permissions\tLinks\tUser\tGroup\tSize\tDate Modified\tName" } { print $1,$2,$3,$4,$5,$6,$7 }'
```

Permissions	Links	User	Group	Size	Date Modified	Name
drwxrwxr-x	3	adrian	adrian	4.0K	03/22	/Applications
drwxrwxr-x	3	adrian	adrian	4.0K	04/02/22	/Calibre
drwxrwxr-x	2	adrian	adrian	4.0K	04/06/22	/challenge-Lab6
drwxr-xr-x	2	adrian	adrian	4.0K	03/01/22	/Desktop
drwxr-xr-x	5	adrian	adrian	4.0K	04/12/22	/Documents
drwxrwxrnx	1	adrian	adrian	28	03/03/22	/dotfiles
drwxr-xr-x	10	adrian	adrian	24K	04/12/22	/Downloads
drwxrwxr-x	2	adrian	adrian	4.0K	04/03/22	/filesAwkExamples
drwxrwxr-x	2	adrian	adrian	4.0K	04/02/22	/games
drwxrwxr-x	27	adrian	adrian	60K	04/02/22	/grdive
drwxrwxr-x	9	adrian	adrian	4.0K	04/02/22	/keep
drwxrwxr-x	2	adrian	adrian	4.0K	04/12/22	/json-files
drwxrwxr-x	7	adrian	adrian	4.0K	04/02/22	/Lab5
-rw-rw-r--	1	adrian	adrian	4.1M	04/02/22	/Lab5.zip
drwxrwxr-x	2	adrian	adrian	4.0K	04/11/22	/Lab6
drwxrwxr-x	2	adrian	adrian	4.0K	04/02/22	/movies
drwxr-xr-x	3	adrian	adrian	4.0K	04/04/22	/Music
drwxr-xr-x	3	adrian	adrian	4.0K	03/05/22	/Pictures
drwxr-xr-x	2	adrian	adrian	4.0K	03/01/22	/Public
-rw-rw-r--	1	adrian	adrian	0.4K	04/02/22	/Song.mp3
drwxrwxr-x	3	adrian	adrian	4.0K	03/02/22	/Templates
drwxrwxr-x	2	adrian	adrian	4.0K	03/01/22	/Templates
-rw-rw-r--	1	adrian	adrian	108	04/09/22	/todo.md
-rwxr-xr--	1	adrian	adrian	511	04/09/22	/urls.sh
drwxr-xr-x	2	adrian	adrian	4.0K	04/07/22	/Videos
drwxrwxr-x	7	adrian	adrian	4.0K	03/21/22	/VirtualBox

1  
7

## The Sed command

The **sed** command is a stream editor that performs operations on files and standard output. For instance, it can search, find and replace, insert, and deletion. By using SED you can edit files without opening them.

- Usage:

- `sed options + sed script + file`
- Basic Example:
  - Replacing a string in given file globally (replace false or true)
    - `sed 's/false/true/g' ~/Documents/sample_files/Json/joke.json`
- More examples:
- Replacing only the fourth occurrence per line in a file:
  - `sed 's/false/true/4' joke.json`
- Replacing from the given number occurrence to the rest occurrences in a file. Start at the second time the word appears and continue to till the end of the file.
  - `sed 's/false/true/' joke.json`
- Replacing string on a specific line number
  - `sed '3 s/false/true/' joke.json`
- Replacing string on a range of lines
  - `sed '1,3 s/false/true/' joke.json`
- Even more examples on sed:
  - To delete a particular line (line 5)
    - `sed '3d' ~/Documents/sample_files/Code/helloworld.py`
  - To delete the last line
    - `sed '$d' ~/Documents/sample_files/Code/helloworld.py`
  - To delete line from range x to y
    - `sed '2,4d' ~/Documents/sample_files/Code/helloworld.py`
  - To delete from a given number to the last line
    - `sed '3,$d' ~/Documents/sample_files/Code/helloworld.py`
  - To delete pattern matching line in a file
    - `sed '/fav/d' ~/Documents/sample_files/Code/helloworld.py`
- Elaborating on Sed:



## Again more on sed commands

- To insert one blank line after each line
  - `sed G helloWorld.c`
- To insert two blank lines
  - `sed 'G;G' helloWorld.c`
- To delete blank lines and insert one blank line after each line
  - `sed '/^$/d;G' helloWorld.c`
- Insert 5 spaces to the left of every lines
  - `sed 's/^/ /' helloWorld.c`

2  
4

How to use | the pipe. How to redirect standard output?

The **pipe (|)** allows you to redirect the standard output of a command to the standard input of another.

- Usage:
  - `command_1 | command_2 | command_3 | ... | command_N`
- Basic examples:
  - Use the grep to look for a string in a particular man page
    - `man ls | grep "human-readable"`
  - Display only the options of the of any command from its man page
    - `man ls | grep "[[:space:]]*[[[:punct:]]"`

Other examples of |:

- Display only the ip addresses from the output of the ip command:
  - `ip addr | grep -Eo '[[[:digit:]]{1,3}\.[[[:digit:]]{1,3}\.[[[:digit:]]{1,3}\.[[[:digit:]]{1,3}'`
- Display only the 2nd line in a file:
  - `head -2 file.lst | tail -1`
- Parse a file with grep and replace a string in the output:
  - `grep -i "honda" cars.csv | sed 's/honda/tesla/g'`

How to save the standard output?

- Usage:
  - `Command output + > + file`
- Basic Examples:

- Save the output of a command to a file
  - `ls -lA ~ > all-files-in-home.txt`
- Save the error generated by a command to a file
  - `ls -lA downloads/ 2> error-of-ls`
- Save the error to a file and the success to another
  - `ls -lA downloads/ Pictures > success.txt 2> error.txt`
- Save the error and success to the same file
  - `ls -lA downloads/ Pictures &> alloutput.txt`
- Do not display errors. Send errors to the black hole
  - `ls -lA downloads/ 2> /dev/null`

## How to append the output of a file

Append means to add more to a file instead of overwriting its content. When we use `>` on a file that already exist and contains data, we overwrite whatever is already inside the file. For example:

- `ls -la > allmyfiles.lst`

In this example, if the file `allmyfiles.lst` had any data prior executing the command, that data will be overwritten by the output of `ls-la`. But, what happens when we don't want to keep the data? Then we use:

- `ls -la >> allmyfiles.lst`

Will add the output of `ls-la` to the end of the file `allmyfiles.lst`