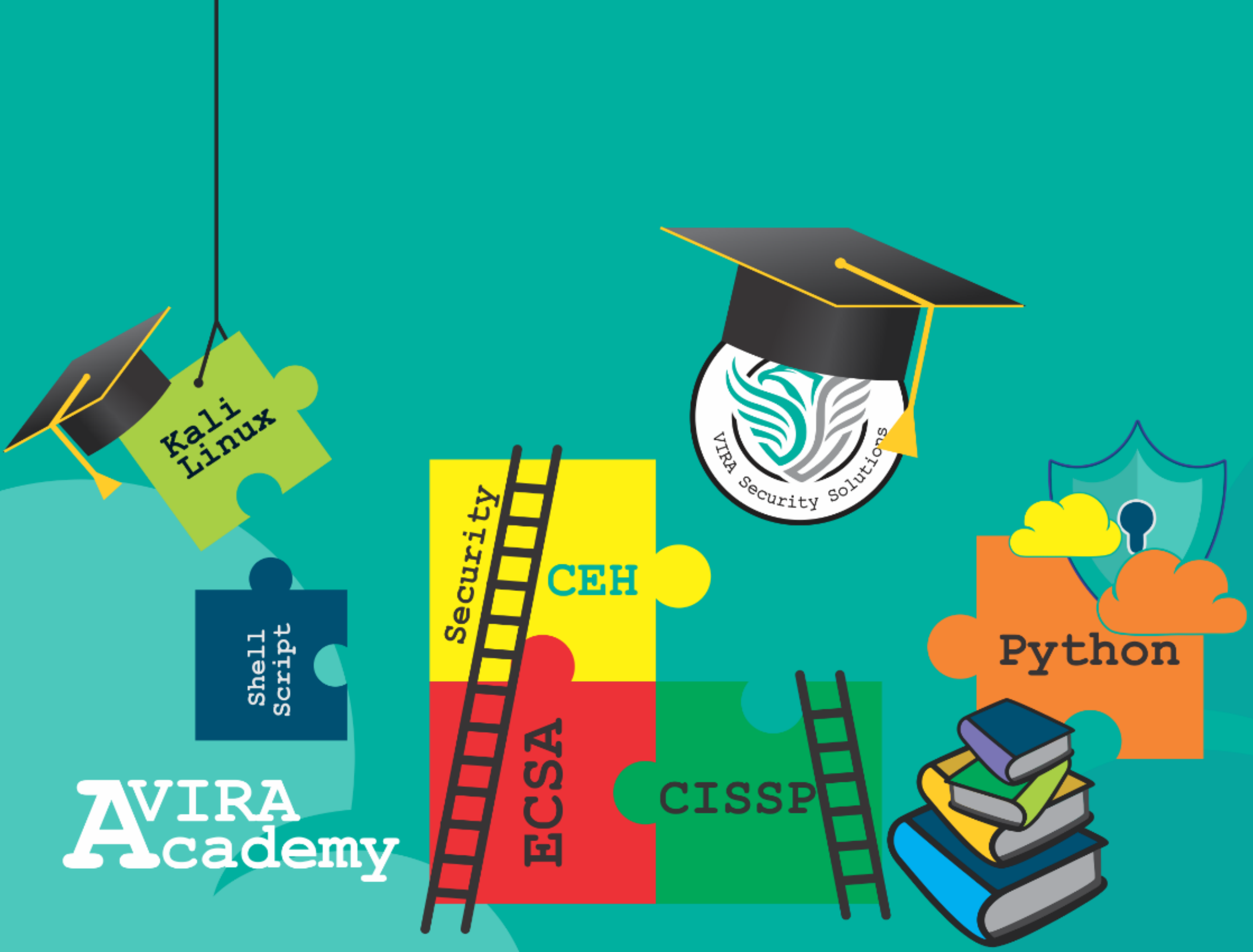

Kali Linux
Shell Script
Security
CEH
ECSA
CISSP
Python
VIRA Security Solutions
AVIRA Academy

Linux

Part3

Mohammad Reza Gerami
mrgerami@aut.ac.ir
gerami@virasecsolutions.com
March 01 2020

Install R from Source

```
curl -O https://cran.rstudio.com/src/base/R-3/R-${R_VERSION}.tar.gz
tar -xzvf R-${R_VERSION}.tar.gz
cd R-${R_VERSION}
```

```
./configure \
  --prefix=/opt/R/${R_VERSION} \
  --enable-memory-profiling \
  --enable-R-shlib \
  --with-blas \
  --with-lapack

make
sudo make install
```

Verify R installation

```
/opt/R/${R_VERSION}/bin/R --version
```

The PATH environment variable

- Colon-separated list of directories.
- Non-absolute pathnames of executables are only executed if found in the list.
  - Searched left to right
- Example:

```
$ myprogram
sh: myprogram not found
$ PATH=/bin:/usr/bin:/home/vira
$ myprogram
hello!
```

Having . In Your Path

```
$ ls
   foo
$ foo
sh: foo: not found
```

```
$ ./foo
Hello, foo.
```

- What **not** to do:

```
$ PATH=.:/bin
$ ls
foo
$ cd /usr/local
$ ls -l

$ ls
```

Shell Variables

- Shells have several mechanisms for creating variables. A variable is a name representing a string value. Example: **PATH**
  - Shell variables can save time and reduce typing errors, variables
- Allow you to store and manipulate information
  - Eg: `ls $DIR > $FILE`
- Two types: **local** and **environmental**
  - *local* are set by the user of by the shell itself
  - *environmental* come from the operating system and are passed to children

Variables (con't)

- Syntax varies by shell
  - `name=value            # sh, ksh`
  - `set name = value   # csh`

- To access the value: **`$var`**

- Turn local variable into environment:

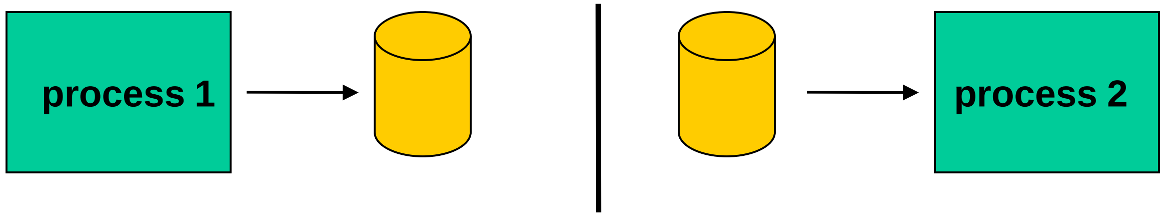
```
export variable
```

Environmental Variables

NAME	MEANING
$HOME	Absolute pathname of your home directory
$PATH	A list of directories to search for
$MAIL	Absolute pathname to mailbox
$USER	Your user id
$SHELL	Absolute pathname of login shell
$TERM	Type of your terminal
$PS1	Prompt

File Approach

- Run first program, save output into file
- Run second program, using file as input



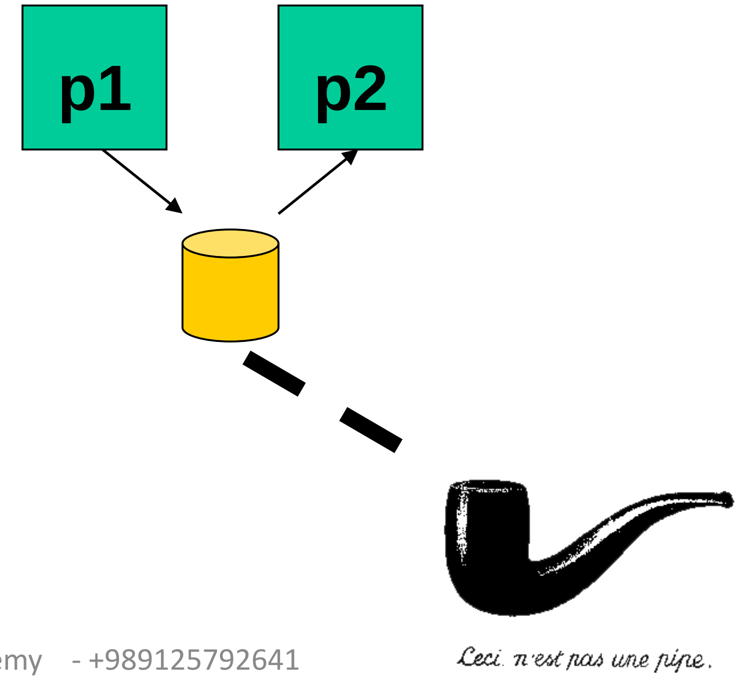
- Unnecessary use of the disk
  - Slower
  - Can take up a lot of space (eg: **ls -R** followed by **wc**)
- Makes no use of multi-tasking

More about pipes

- What if a process tries to read data but nothing is available?
  - UNIX puts the reader to sleep until data available
- What if a process can't keep up reading from the process that's writing?
  - UNIX keeps a buffer of unread data
    - This is referred to as the *pipe size*.
  - If the pipe fills up, UNIX puts the writer to sleep until the reader frees up space (by doing a read)
- Multiple readers and writers possible with pipes.

Interprocess Communication For Unrelated Processes

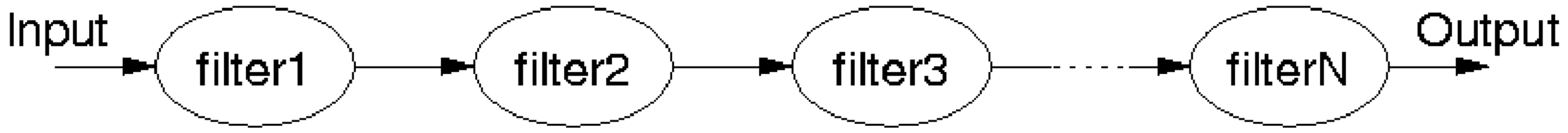
- FIFO (*named pipes*)
  - A special file that when opened represents pipe
- System V IPC
  - message queues
  - semaphores
  - shared memory
- Sockets (client/server model)



Shell Pipelines

- Output of one program becomes input to another
  - Uses concept of UNIX **pipes**
- Example: **`$ who | wc -l`**
  - counts the number of users logged in
- Pipelines can be long

```
filter1 | filter2 | filter3 | ... | filterN
```



What’s the difference?

Both of these commands send input to ***command*** from a file instead of the terminal:

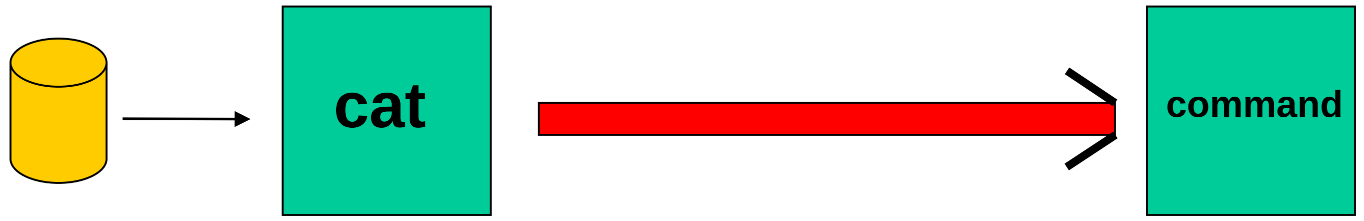
```
$ cat file | command
```

vs.

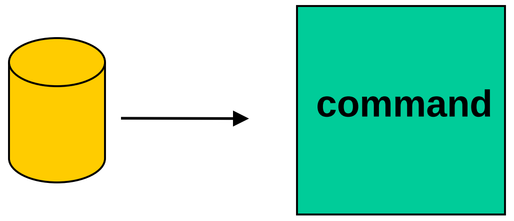
```
$ command < file
```

An Extra Process

```
$ cat file | command
```



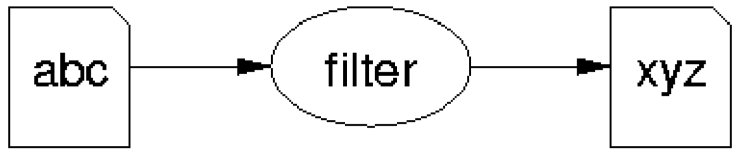
```
$ command < file
```



Introduction to Filters

- A class of Unix tools called ***filters***.
  - Utilities that read from standard input, transform the file, and write to standard out
- Using filters can be thought of as *data oriented programming*.
  - Each step of the computation transforms data *stream*.

```
filter < abc > xyz
```



Examples of Filters

- **Sort**
  - Input: lines from a file
  - Output: lines from the file sorted
- **Grep**
  - Input: lines from a file
  - Output: lines that match the argument
- **Awk**
  - Programmable filter

cat: The simplest filter

- The cat command copies its input to output unchanged (*identity filter*). When supplied a list of file names, it con**cat**enates them onto stdout.
- Some options:
  - **`-n`** **n**umber output lines (starting from 1)
  - **`-v`** display control-characters in **v**isible form (e.g. ^C)

```
cat file*

ls | cat -n
```

head

- Display the first few lines of a specified file
- Syntax: *head [-n] [filename...]*
  - *-n* - number of lines to display, default is 10
  - *filename...* - list of filenames to display
- When more than one filename is specified, the start of each files listing displays

```
==>filename<==
```

tail

- Displays the last part of a file
- Syntax: *tail +|-number [lbc] [f] [filename]*

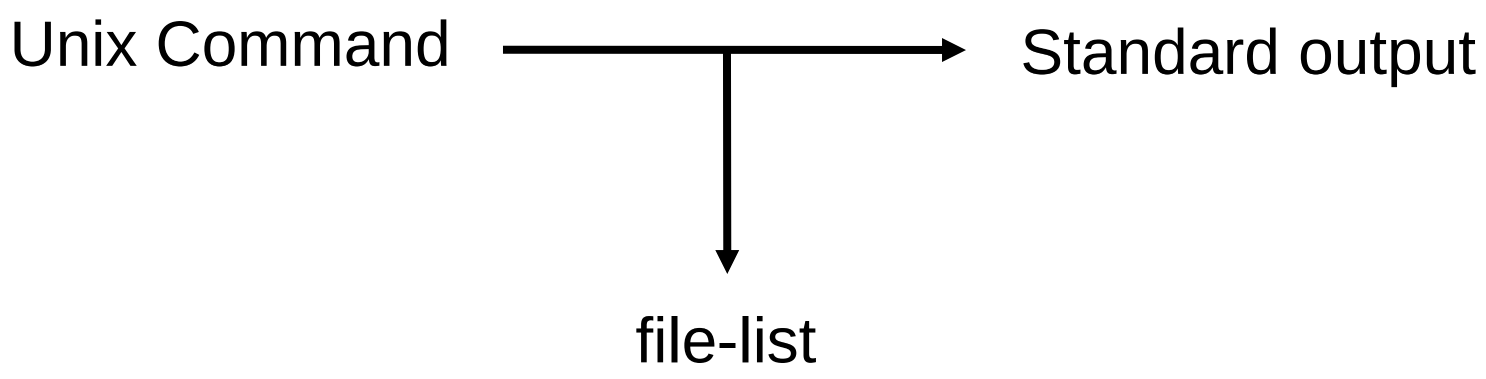
  or: *tail +|-number [l] [rf] [filename]*

  – *+number* - begins copying at distance *number* from beginning of file, if *number* isn't given, defaults to 10
  – *-number* - begins from end of file
  – *l,b,c* - *number* is in units of lines/block/characters
  – *r* - print in reverse order (lines only)
  – *f* - if input is not a pipe, do not terminate after end of file has been copied but loop. This is useful to monitor a file being written by another process

head and tail examples

```
head /etc/passwd
head *.c
tail +20 /etc/passwd
ls -lt | tail -3
head –100 /etc/passwd | tail -5
tail –f /usr/local/httpd/access_log
```

tee



- Copy standard input to standard output and one or more files
  - Captures intermediate results from a filter in the pipeline

tee con’t

- Syntax: *tee [-ai] file-list*
  - *-a* - append to output file rather than overwrite, default is to overwrite (replace) the output file
  - *-i* - ignore interrupts
  - *file-list* - one or more file names for capturing output
- Examples

```
ls | head –10 | tee first_10 | tail –5

who | tee user_list | wc
```

Unix Text Files: Delimited Data

Tab Separated

```
John      99
Anne      75
Andrew    50
Tim       95
Arun      33
Sowmya    76
```

Pipe-separated

```
COMP1011|2252424|Abbot, Andrew John |3727|1|M
COMP2011|2211222|Abdurjh, Saeed |3640|2|M
COMP1011|2250631|Accent, Aac-Ek-Murhg |3640|1|M
COMP1021|2250127|Addison, Blair |3971|1|F
COMP4012|2190705|Allen, David Peter |3645|4|M
COMP4910|2190705|Allen, David Pater |3645|4|M
```

Colon-separated

```
root:ZHolHAHZw8As2:0:0:root:/root:/bin/ksh
vira:nJz3ru5a/44Ko:100:100:Vira:/home/vira:/bin/bash
cs1021:iZ3sO90O5eZY6:101:101:COMP1021:/home/cs1021:/bin/bash
cs2041:rX9KwSSPqkLyA:102:102:COMP2041:/home/cs2041:/bin/csh
cs3311:mLRiCIvmtI9O2:103:103:COMP3311:/home/cs3311:/bin/sh
```

cut: select columns

- The cut command prints selected parts of input lines.
  - can select columns (assumes tab-separated input)
  - can select a range of character positions
- Some options:
  - **-f** *listOfCols*: print only the specified columns (tab-separated) on output
  - **-c** *listOfPos*: print only chars in the specified positions
  - **-d** *c*: use character *c* as the column separator
- Lists are specified as ranges (e.g. 1-5) or comma-separated (e.g. 2,4,5).

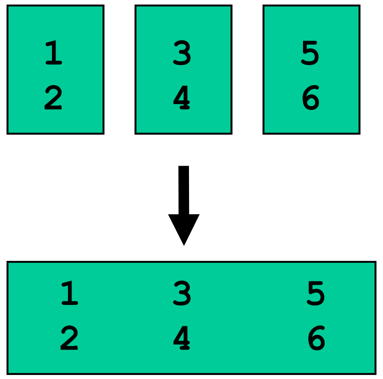
cut examples

```
cut -f 1 < data

cut -f 1-3 < data

cut -f 1,4 < data

cut -f 4- < data

cut -d'|' -f 1-3 < data

cut -c 1-4 < data
```

Unfortunately, there's no way to refer to "last column" without counting the columns.

paste: join columns

- The paste command displays several text files "in parallel" on output.
- If the inputs are files **a**, **b**, **c**
  - the first line of output is composed of the first lines of **a**, **b**, **c**
  - the second line of output is composed of the second lines of **a**, **b**, **c**
- Lines from each file are separated by a tab character.



- If files are different lengths, output has all lines from longest file, with empty strings for missing lines.

paste example

```
cut -f 1 < data > data1
cut -f 2 < data > data2
cut -f 3 < data > data3
paste data1 data3 data2 > newdata
```

sort: Sort lines of a file

- The sort command copies input to output but ensures that the output is arranged in ascending order of lines.
  - By default, sorting is based on ASCII comparisons of the whole line.
- Other features of sort:
  - understands text data that occurs in columns. (can also sort on a column other than the first)
  - can distinguish numbers and sort appropriately
  - can sort files "in place" as well as behaving like a filter
  - capable of sorting *very large* files

sort: Options

- Syntax: *sort [-dftnr] [-o filename] [filename(s)]*

-d Dictionary order, only letters, digits, and whitespace are significant in determining sort order

-f Ignore case (fold into lower case)

-t Specify delimiter

-n Numeric order, sort by arithmetic value instead of first digit

-r Sort in reverse order

-o filename - write output to filename, filename can be the same as one of the input files

- Lots of more options…

sort: Specifying fields

- Delimiter : **`-t`***d*
- Old way:
  - **+***f***[.***c***] [***options***] [-***f***[.***c***] [***options***]**
    - **`+2.1 -3 +0 -2 +3n`**
  - Exclusive
  - Start from 0 (unlike cut, which starts at 1)
- New way:
  - **-k** *f***[.***c***] [***options***] [,***f***[.***c***] [***options***]]**
    - **`-k2.1 -k0,1 -k3n`**
  - Inclusive
  - Start from 1

sort Examples

```
sort +2nr < data
sort -k2nr data

sort -t: +4 /etc/passwd

sort -o mydata mydata
```

uniq: list UNIQue items

- Remove or report adjacent duplicate lines
- Syntax: *uniq [-cdu] [input-file] [output-file]*

-c Supersede the -u and -d options and generate an output report with each line preceded by an occurrence count

-d Write only the duplicated lines

-u Write only those lines which are not duplicated

  – The default output is the union (combination) of *-d* and *-u*

wc: Counting results

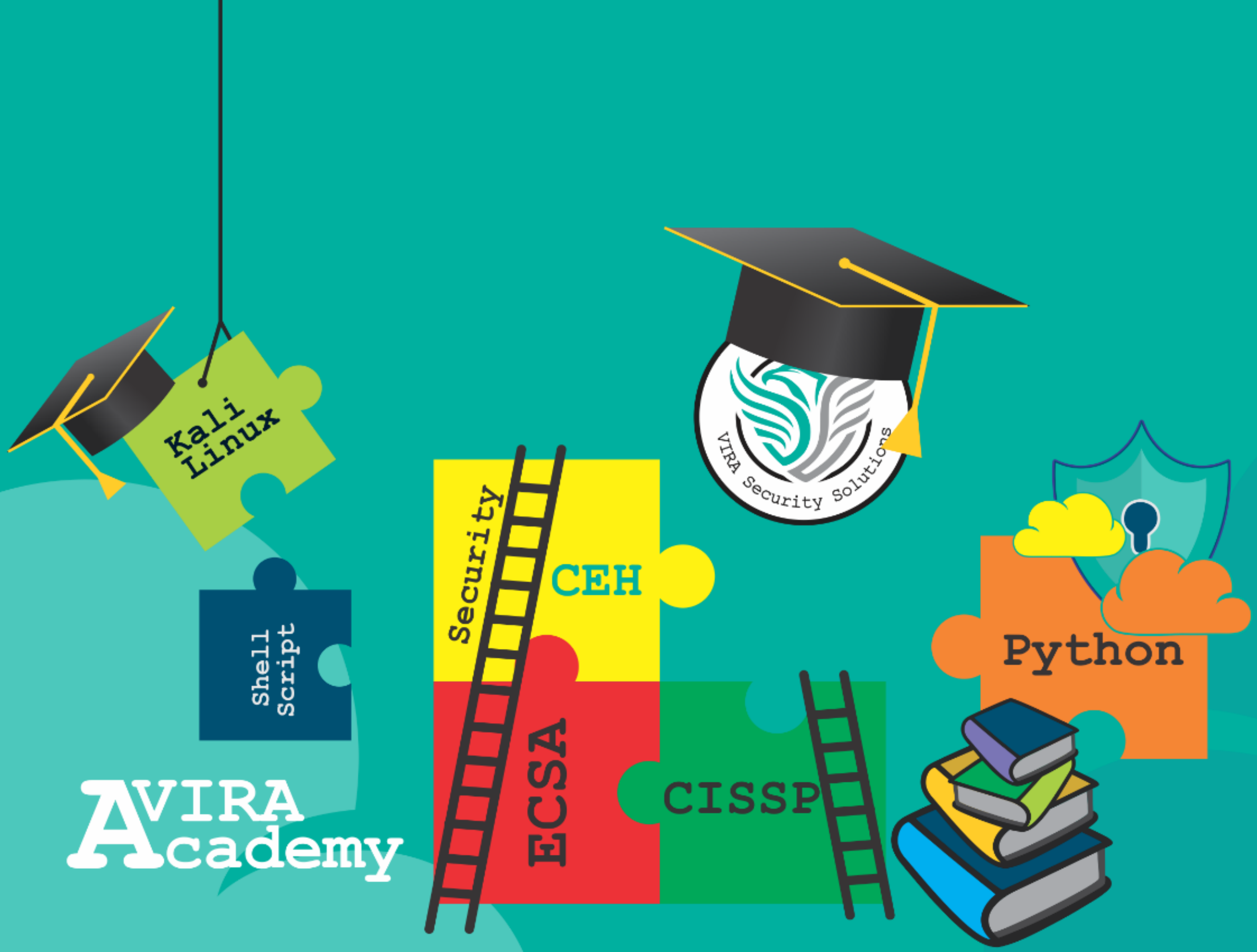
- The word count utility, **wc**, counts the number of lines, characters or words
- Options:

`-l`	Count lines
`-w`	Count words
`-c`	Count characters

- Default: count lines, words and chars

wc and uniq Examples

```
who | sort | uniq -d
wc my_essay
who | wc
sort file | uniq | wc -l
sort file | uniq -d | wc -l
sort file | uniq -u | wc -l
```


Kali Linux
Shell Script
Security
CEH
ECSA
CISSP
Python
VIRA Security Solutions
AVIRA Academy