Current Location – pwd

Create a directory – mkdir <Folder Name> (mkdir Student)

Change Directory – cd <Folder name>

Return back – cd ..

Create a file – touch <file name> (touch index.txt)

See all files in current directory – ls

More ls commands

-a: Display all files

-b: Displays non printing characters in octal

-c: Display files by file timestamp

-C: Display files in columnar format (default)

-d: Displays only directories

-l: Displays long format listing

-L: Display file or directory referenced by a symbolic link

-m: Display names as a comma separated list

-n: Displays long format listing with GID and UID numbers

-o: Displays long format listing but excludes group names

-p: Displays directories with /

-q: Displays all non printing characters as ?

-r: Display files in reverse order

-R: Displays subdirectories as well

-t: Display newest files first (Based on timestamp)

-u: Display files by the access time

-x: Display files as rows across the screen

-1: Display each entry on a line

Output file in a window – cat <file name> or more <file name> or less<file name> for open a file seperately

Create file in command line editor – vi <file name>

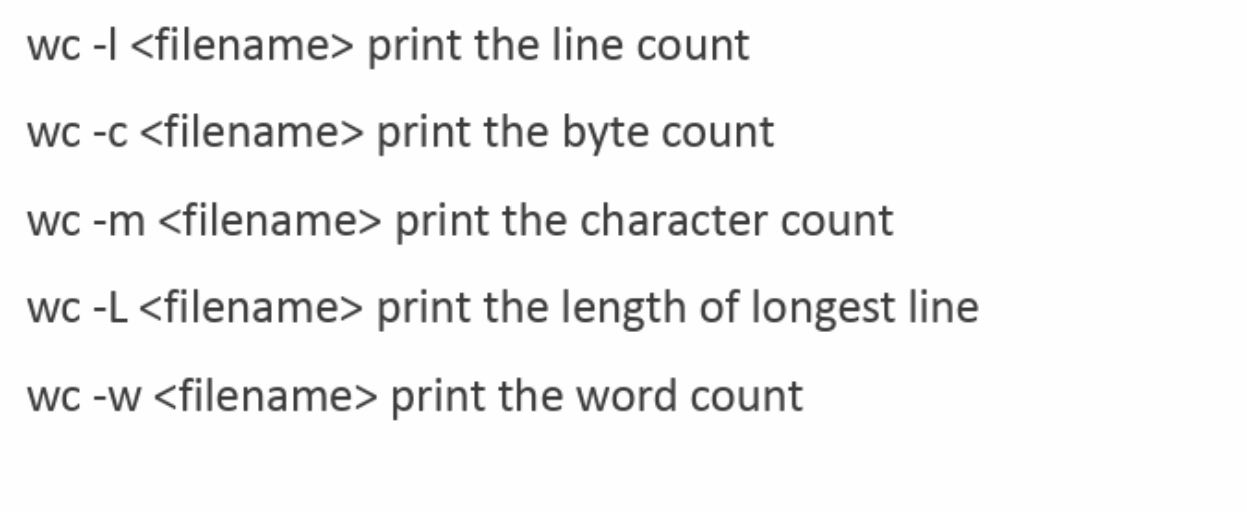
Write something: ESC + i

Save: ESC + : + wq

Save and quit : ESC + : + wq

Quit without save: ESC + : + q!

wc command: Gives the number of lines, words and letters of a file (wc <filename>)



mv command: Send data from one file to another file (mv file1.txt file2.txt)

cp command: Copy data from one file to another

rm: Remove a file from current directory (rm <filename>)

rmdir: Remove a directory. Directory has to be empty (rm <directory name>

rm -R: Remove a directory which include files (rm -R < directory name>)

history: Show command history

ifconfig: Show machine's IP address

ping: ping a domain (ping google.com)

Who:

This is used to find out who's working on our system. As we now know, Linux is a multiuser system. Even if we're using one computer at our home, we may be working as more than one person. For example, if we logged in as 'root' but are working as 'guest-hedbs3'. We may see something like this:

whoami: Know who you are

ps: List of processes running on our system

uname -a: Summary about the system

chmod:

mode:

Who u=user, g=group, o=other, a=all (default)

Opcode

+ means add permission

— means remove permission

= means assign permission and remove the permission of unspecified fields

Permission r=Read, w=write, x=Execute

Examples:

chmod ug+rw mydir

chmod a-w myfile

VARIABLES

User defined variables =>

- Null UDV => var1 = / var1 = ‘’
- With value => var1=2
- Print a variable => echo $var1 (without $, it will print as var1)

System defined variables

System Defined Variables	Meaning
BASH=/bin/bash	Shell Name
BASH_VERSION=4.1.2(1)	Bash Version
COLUMNS=80	No. of columns for our screen
HOME=/home/linuxtechi	Home Directory of the User
LINES=25	No. of columns for our screen
LOGNAME=LinuxTechi	LinuxTechi Our logging name
OSTYPE=Linux	OS type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Path Settings
PS1=[\u@\h \W]\$	Prompt Settings
PWD=/home/linuxtechi	Current Working Directory
SHELL=/bin/bash	Shell Name
USERNAME=linuxtechi	User name who is currently login to system

Arithmetic Expressions

- echo $((10+5))
- expr 1 + 3 (space between characters is needed)

SHELL

- Create a shell -> nano test.sh (ctrl+x to save) (ls again to check it is saved)
- To execute –
  - chmod +x test.sh (Give execute permissions)
  - ./test.sh (This will output the shell content)
  - In a shell ->
    - Get user input -> read -p "Enter 2 numbers : " x y (When running, 'Enter 2 numbers :' will be displayed then inputs will be given space separated in thevar same line)
    - ++var1 , var1++ are also working in linux. Var1**var2 gives var1^var2
    - Another example: Get 3 strings
      read string1 string2 string3 (Inputs should be given space separated)
    - Get inputs as an array
      read -a names (Input should be given as space separated. echo $names will print array)

Mathematical Operator in Shell Script	Meaning	Normal Mathematical Statements	But in Shell	
			For test statement with if command	For [expr] statement with if command
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

- In shell,
  If [$var1 -eq <some_value>]
  then
  <what to do If true>
  else
  <what to do if false>
  Fi

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

(In above table, for equality, it is ==, not =)

ASCII Comparison

```
x=c
if [[ $x < "b" ]]
then
echo "yes"
else
echo "no"
fi
```

Using else-if

```
word="a"
if [[ $word == "b" ]]
then
echo "condition b is true"
elif [[ $word == "a" ]]
then
echo "condition a is true"
else
echo "condition is false"
```

List all items in the current working directory

```
for item in *
do
echo $item
done
```

List only files. Not directories. For directories, use -d instead of -f

```
for item in *
do
if [ -f $item ]
then
echo $item
fi
done
```

While loop

```
x=0
while [ $x -le 10 ]
do
echo $x
((x++))
done
```

For loop variation

```
for i in 1 2 4 5 7 10
do
echo "Welocme $i times"
done
```

This will print the string with only numbers in the above list

Switch-Case

```
x=10
case $x in
5) echo "5";;
6) echo "6";;
*) echo "else";;
esac
```

This can be used for strings in the same way too

Create a file and add data -> cat > <filename>, then add data be pressing enter one by one, then ctrl+d to save

Sort content of a file -> sort <filename>

Reverse content -> sort -r <filename>

Options	Meaning
-b	Ignores leading spaces and tabs.
-c	Checks if files are already sorted. If they are, sort does nothing.
-d	Sorts in dictionary order (ignores punctuation).
-f	Ignores case
-m	Merges files that have already been sorted.
-n	Sorts in numeric order.
-ofile	Stores output in file.
-r	Reverses sort
-tc	Separates fields with character (default is tab).

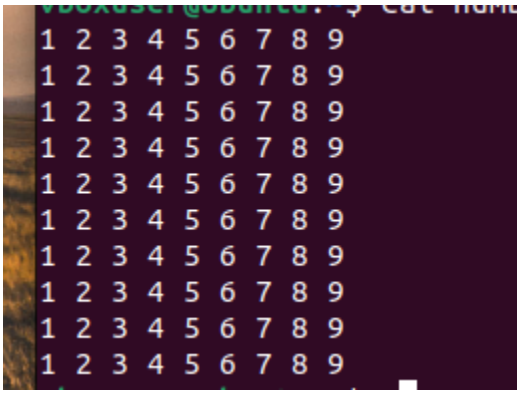
SUB-STRING COMMAND (CUT)

- echo I like Cricket | cut -b 1 (This will output first byte, that means 'I'. If we replace any index of a character of the sentence it will return that character)

Create a file of numbers

```
vboxuser@Ubuntu:~$ for i in $(seq 10); do echo $(seq -s ' ' 1 9) >> number.txt;
done
```

This means, create 10 lines and each line will have numbers from 1 to 9 separated by a space . We can use any character as the separator instead of space
('done' is also a part of the command)

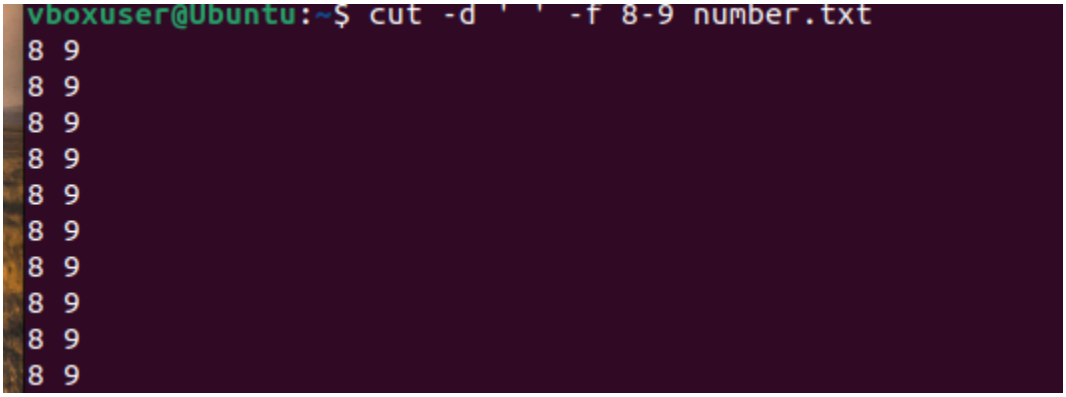


← Output

Filter the above file

```
vboxuser@Ubuntu:~$ cut -d ' ' -f 2 number.txt
2
2
2
2
2
2
2
2
2
2
vboxuser@Ubuntu:~$ cut -d ' ' -f 8 number.txt
8
8
8
8
8
8
8
8
8
8
vboxuser@Ubuntu:~$
```

- -d ‘ ‘ -> This means the separator of each field. In this case, it is ‘space’
- -f 2 -> This means the field number
- number.txt is the file name
- **Range of fields**



- **Only selected fields**

```
vboxuser@Ubuntu:~$ cut -d ' ' -f 8,9,1 number.txt
1 8 9
1 8 9
1 8 9
1 8 9
1 8 9
1 8 9
1 8 9
1 8 9
1 8 9
1 8 9
```

- **Combination of filters – Until 2nd row, eliminate 3rd row, again 4-6, eliminate 7 and again from 8 to end**

```
vboxuser@Ubuntu:~$ cut -d ' ' -f -2,4-6,8- number.txt
1 2 4 5 6 8 9
1 2 4 5 6 8 9
1 2 4 5 6 8 9
1 2 4 5 6 8 9
1 2 4 5 6 8 9
1 2 4 5 6 8 9
1 2 4 5 6 8 9
1 2 4 5 6 8 9
1 2 4 5 6 8 9
1 2 4 5 6 8 9
```

- **Eliminate selected rows and get the output**

```
vboxuser@Ubuntu:~$ cut -d ' ' -f 3 --complement number.txt
1 2 4 5 6 7 8 9
1 2 4 5 6 7 8 9
1 2 4 5 6 7 8 9
1 2 4 5 6 7 8 9
1 2 4 5 6 7 8 9
1 2 4 5 6 7 8 9
1 2 4 5 6 7 8 9
1 2 4 5 6 7 8 9
1 2 4 5 6 7 8 9
1 2 4 5 6 7 8 9
vboxuser@Ubuntu:~$ cut -d ' ' -f 3-5 --complement number.txt
1 2 6 7 8 9
1 2 6 7 8 9
1 2 6 7 8 9
1 2 6 7 8 9
1 2 6 7 8 9
1 2 6 7 8 9
1 2 6 7 8 9
1 2 6 7 8 9
1 2 6 7 8 9
1 2 6 7 8 9
```

- **Add more lines to file**

```
vboxuser@Ubuntu:~$ echo 123456789 >> number.txt
```

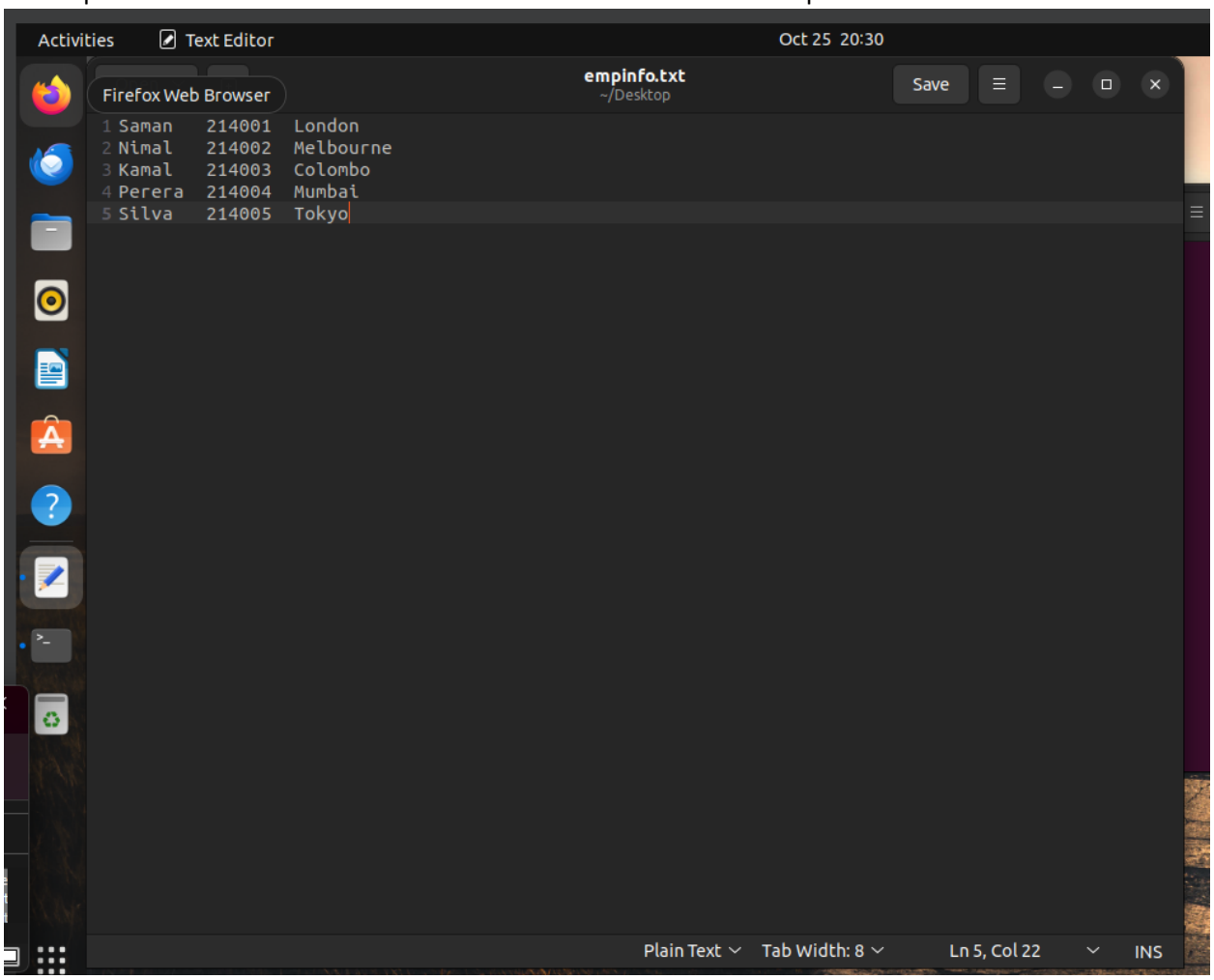
- So, it will add a different field without space separator. So, now if we output the file field using above command, for every field numbers, it will return the above unique line

- But with '-s' flag, we can remove lines without the space separator

```
vboxuser@Ubuntu:~$ cut -s -d ' ' -f 8 number.txt
8
8
8
8
8
8
8
8
8
8
```

DATABASE HANDLING

- You can create a file in the current working directory using touch <file name> (Use .txt extension)
- Then open the file from the text editor and add dataset. Use tab to separate columns



In above file, columns are separated using tabs and likewise, OS identifies them as columns automatically. We can use 'nano <filename> too to add data.

- Now we can use filter commands for the database like this

Cut Command

```
vboxuser@Ubuntu:~/Desktop$ cut -f 1,3 empinfo.txt
Saman   London
Nimal   Melbourne
Kamal   Colombo
Perera  Mumbai
Silva   Tokyo
```

Sort Command

```
vboxuser@Ubuntu:~/Desktop$ sort empinfo.txt
Kamal   214003  Colombo
Nimal   214002  Melbourne
Perera  214004  Mumbai
Saman   214001  London
Silva   214005  Tokyo
```

The cut command assumes that the fields are separated by tab character. If the fields are delimited by some character other than the default tab character, cut supports an option –d which allows us to set delimiter. The file *empinfo* may have the information for each employee stored in the following format.

```
Name: age: address: city: pin: division
```

File ->

```
1 Saman:214001:London
2 Nimal:214002:Melbourne
3 Kamal:214003:Colombo
4 Perera:214004:Mumbai
5 Silva:214005:Tokyo
```

Filter ouput->

```
vboxuser@Ubuntu:~/Desktop$ cut -d ':' -f 1,3 emp2.txt
Saman:London
Nimal:Melbourne
Kamal:Colombo
Perera:Mumbai
Silva:Tokyo
```

Cut command with -c flag : It will identify each character as a column and outputs character columns

```
vboxuser@Ubuntu:~/Desktop$ cut -c 1-2 empinfo.txt
Sa
Ni
Ka
Pe
Si
vboxuser@Ubuntu:~/Desktop$ cut -c 1-10 empinfo.txt
Saman    2140
Nimal    2140
Kamal    2140
Perera   214
Silva    2140
```

Find a word in a file (grep command)

```
vboxuser@Ubuntu:~/Desktop$ grep love perfect.txt
I found a love, for me
'Cause we were just kids when we fell in love
I found a lover, to carry more than just my secrets
To carry love, to carry children of our own
We are still kids, but we're so in love
vboxuser@Ubuntu:~/Desktop$ grep perfect perfect.txt
Darling, you look perfect tonight
Darling, you look perfect tonight
And she looks perfect
You look perfect tonight
vboxuser@Ubuntu:~/Desktop$ grep deserve perfect.txt
I don't deserve this
I don't deserve this
vboxuser@Ubuntu:~/Desktop$ grep person perfect.txt
Now I know I have met an angel in person
```

```
vboxuser@Ubuntu:~/Desktop$ grep Saman empinfo.txt
Saman    214001  London
vboxuser@Ubuntu:~/Desktop$ grep 214 empinfo.txt
Saman    214001  London
Nimal    214002  Melbourne
Kamal    214003  Colombo
Perera   214004  Mumbai
Silva    214005  Tokyo
```

```
vboxuser@Ubuntu:~/Desktop$ grep ' a ' perfect.txt
I found a love, for me
Well, I found a girl, beautiful and sweet
When you said you looked a mess
Well, I found a woman, stronger than anyone I know
I found a lover, to carry more than just my secrets
```

-i: Remove case sensitivity

```
vboxuser@Ubuntu:~/Desktop$ grep -i Saman empinfo.txt
Saman   214001  London
saman   214001  London
```

(I added a row as saman instead of Saman to the file)

-n: Get line numbers

```
vboxuser@Ubuntu:~/Desktop$ grep -n love perfect.txt
1:I found a love, for me
5:'Cause we were just kids when we fell in love
21:I found a lover, to carry more than just my secrets
22:To carry love, to carry children of our own
23:We are still kids, but we're so in love
```

-v: Returns the lines without the given parameter

```
vboxuser@Ubuntu:~/Desktop$ grep -n -i -v Saman empinfo.txt
3:Nimal 214002  Melbourne
4:Kamal 214003  Colombo
5:Perera        214004  Mumbai
6:Silva 214005  Tokyo
```

(It returns the lines without the word Saman)

-c: Count the lines of the filter

```
vboxuser@Ubuntu:~/Desktop$ grep -c love perfect.txt
5
vboxuser@Ubuntu:~/Desktop$ grep -c -i saman empinfo.txt
2
```

AWK

Some of the key features of Awk are:

- Awk views a text file as records and fields.
- Like common programming language, Awk has variables, conditionals and loops
- Awk has arithmetic and string operators.
- Awk can generate formatted reports
- Awk reads from a file or from its standard input, and outputs to its standard output.
- Awk does not get along with non-text files.

Syntax:

```
awk '/search pattern1/ {Actions} /search pattern2/ {Actions}' file_name
```

In the above awk syntax:

- search pattern is a regular expression.
- Actions – statement(s) to be performed.
- several patterns and actions are possible in Awk.
- Filename – Input file.
- Single quotes around program is to avoid shell not to interpret any of its special characters.

Awk Working Methodology

1. Awk reads the input files one line at a time.
2. For each line, it matches with given pattern in the given order, if matches performs the corresponding action.
3. If no pattern matches, no action will be performed.
4. In the above syntax, either search pattern or action are optional, But not both.
5. If the search pattern is not given, then Awk performs the given actions for each line of the input.
6. If the action is not given, print all that lines that matches with the given patterns which is the default action.
7. Empty braces without any action does nothing. It won't perform default printing operation.
8. Each statement in Actions should be delimited by semicolon.

- Create a file -> touch ex.txt
- Open the file in the terminal to edit -> nano ex.txt
- Type,

  This is a test

This is a second test
This is a third test
This is a fourth test
This is a fifth test

Then ctrl + x, then ‘y’, then ‘enter’ to save

- You can use the ubuntu text editor to add the above content to the file too.
- Then, to get each field’s content (Fields are separated by a space as default), awk ‘{print $0}’ ex.txt

```
vboxuser@Ubuntu:~/Desktop$ awk '{print $0}' ex.txt
This is a test
This is a second test
This is a third test
This is a fourth test
This is a fifth test
```

It prints all because $0 for whole line

- $0 for the whole line.
- $1 for the first field.
- $2 for the second field.
- $n for the nth field.

```
vboxuser@Ubuntu:~/Desktop$ awk '{print $1}' ex.txt
This
This
This
This
This
vboxuser@Ubuntu:~/Desktop$ awk '{print $2}' ex.txt
is
is
is
is
is
```

- Use ‘NR’ for line number

```
vboxuser@Ubuntu:~/Desktop$ awk '{print NR, $0}' ex.txt
1 This is a test
2 This is a second test
3 This is a third test
4 This is a fourth test
5 This is a fifth test
```

- Output field separator (OFS)

```
vboxuser@Ubuntu:~/Desktop$ date | awk ' OFS="-" {print $2,$3,$6}'
Oct-25-MST
vboxuser@Ubuntu:~/Desktop$ date | awk ' OFS="/" {print $2,$3,$6}'
Oct/25/MST
vboxuser@Ubuntu:~/Desktop$ date | awk ' OFS=" " {print $2,$3,$6}'
Oct 25 MST
vboxuser@Ubuntu:~/Desktop$ date | awk ' {print $2,$3,$6}'
Oct 25 MST
```

Default OFS is space

NR: NR command keeps a current count of the number of input records. Remember that records are usually lines. Awk command performs the pattern/action statements once for each record in a file.
NF: NF command keeps a count of the number of fields within the current input record.
FS: FS command contains the field separator character which is used to divide fields on the input line. The default is "white space", meaning space and tab characters. FS can bereassigned to another character (typically in BEGIN) to change the field separator.
RS: RS command stores the current record separator character. Since, by default, an input line is the input record, the default record separator character is a newline.
OFS: OFS command stores the output field separator, which separates the fields when Awk prints them. The default is a blank space. Whenever print has several parameters separated with commas, it will print the value of OFS in between each parameter.
ORS: ORS command stores the output record separator, which separates the output lines when Awk prints them. The default is a newline character. print automatically outputs the contents of ORS at the end of whatever it is given to print.
To print whole file then you can use below command,