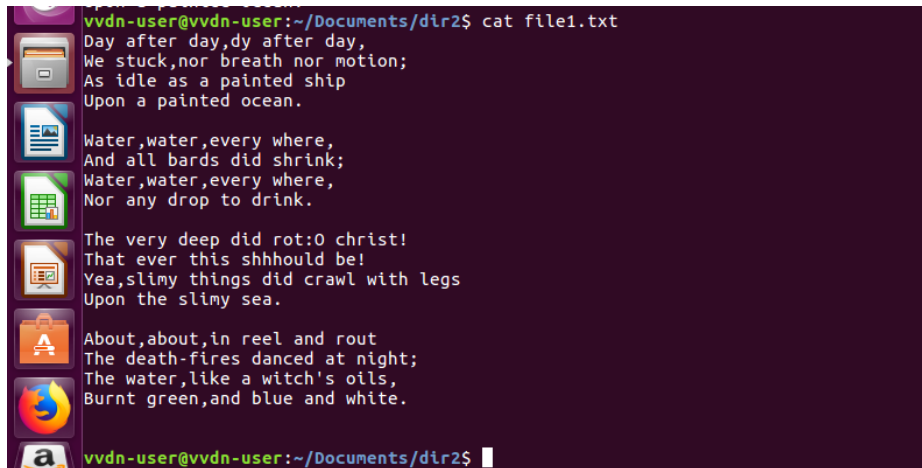


AWK AND SED COMMANDS

SED COMMANDS:

sed is a stream editor that works on piped input or files of text. It doesn't have an interactive text editor interface, however. Rather, you provide instructions for it to follow as it works through the text.



```
vvdn-user@vvdn-user:~/Documents/dir2$ cat file1.txt
Day after day,dy after day,
We stuck,nor breath nor motion;
As idle as a painted ship
Upon a painted ocean.

Water,water,every where,
And all bards did shrink;
Water,water,every where,
Nor any drop to drink.

The very deep did rot:O christ!
That ever this shhould be!
Yea,slimy things did crawl with legs
Upon the slimy sea.

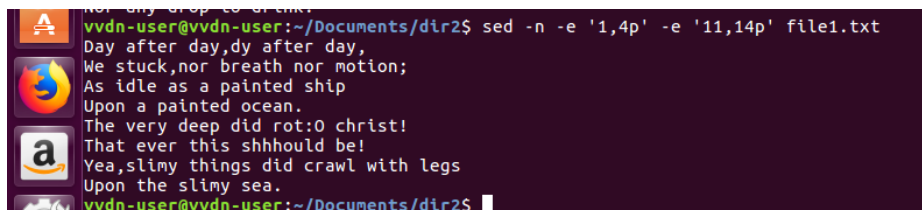
About,about,in reel and rout
The death-fires danced at night;
The water,like a witch's oils,
Burnt green,and blue and white.
```

To select some lines from the file, we provide the start and end lines of the range we want to select.



```
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n '1,4p' file1.txt
Day after day,dy after day,
We stuck,nor breath nor motion;
As idle as a painted ship
Upon a painted ocean.
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n '6,9p' file1.txt
Water,water,every where,
And all bards did shrink;
Water,water,every where,
Nor any drop to drink.
```

Use the -e (expression) option to make multiple selections.



```
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n -e '1,4p' -e '11,14p' file1.txt
Day after day,dy after day,
We stuck,nor breath nor motion;
As idle as a painted ship
Upon a painted ocean.
The very deep did rot:O christ!
That ever this shhould be!
Yea,slimy things did crawl with legs
Upon the slimy sea.
```

Choose a starting line and tell sed to step through the file and print alternate lines, every fifth line, or to skip any number of lines.

```
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n '1~2p' file1.txt
Day after day,dy after day,
As idle as a painted ship

And all bards did shrink;
Nor any drop to drink.
The very deep did rot:O christ!
Yea,slimy things did crawl with legs

The death-fires danced at night;
Burnt green,and blue and white.
vvdn-user@vvdn-user:~/Documents/dir2$
```

The caret (^) represents the start of the line. We'll enclose our search term in forward slashes (/). We also include a space after "And" so words like "Android" won't be included in the result.

```
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n '/^And/p' file1.txt
And all bards did shrink;
vvdn-user@vvdn-user:~/Documents/dir2$
```

The s tells sed this is a substitution. The first string is the search pattern, and the second is the text with which we want to replace that matched text.

```
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n 's/day/week/p' file1.txt
Day after week,dy after day,
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n 's/day/week/p' file1.txt
Day after week,day after day,
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n 's/day/week/gp' file1.txt
Day after week,week after week,
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n 's/day/week/gip' file1.txt
week after week,week after week,
vvdn-user@vvdn-user:~/Documents/dir2$
```

The search pattern is space, space asterisk (*), and the substitution string is a single space. The 1,4 restricts the substitution to the first four lines of the file.

```
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n '1,4p' file1.txt
Day after day,day after day,
We stuck,nor breath nor motion;
As idle as a painted ship
Upon a painted ocean.
vvdn-user@vvdn-user:~/Documents/dir2$ sed -n '1,4 s/ */ /gp' file1.txt
Day after day , day after day ,
We stuck , nor breath nor motion ;
As idle as a painted ship
Upon a painted ocean .
vvdn-user@vvdn-user:~/Documents/dir2$
```

Another example,

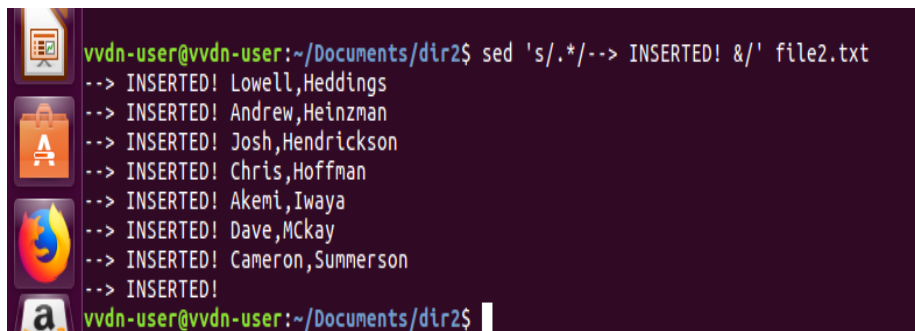
```
vvdn-user@vvdn-user:~/Documents/dir2$ cat file2.txt
Lowell,Heddings
Andrew,Heinzman
Josh,Hendrickson
Chris,Hoffman
Akemi,Iwaya
Dave,Mckay
Cameron,Summerson
```

- **sed 's/:** The normal substitution command.
- **^:** Because the caret isn't in a group ([]), it means "The start of the line."
- **\(.*\),:** The first subexpression is any number of any characters. It's enclosed in parentheses ([]), each of which is preceded by a backslash (\) so we can reference it by number. Our entire search pattern so far translates as search from the start of the line up to the first comma (,) for any number of any characters.
- **\(.*\):** The next subexpression is (again) any number of any character. It's also enclosed in parentheses ([]), both of which are preceded by a backslash (\) so we can reference the matching text by number.
- **\$/:** The dollar sign (\$) represents the end of the line and will allow our search to continue to the end of the line. We've used this simply to introduce the dollar sign. We don't really need it here, as the asterisk (*) would go to the end of the line in this scenario. The forward slash (/) completes the search pattern section.
- **\2,\1 /g':** Because we enclosed our two subexpressions in parentheses, we can refer to both of them by their numbers. Because we want to reverse the order, we type them as second-match,first-match. The numbers have to be preceded by a backslash (\).
- **/g:** This enables our command to work globally on each line.



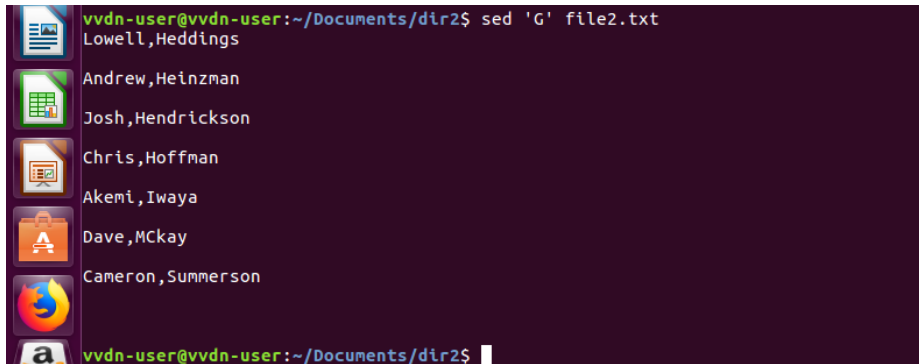
```
vvdn-user@vvdn-user:~/Documents/dir2$ sed 's/^(.*),\(.*\)$/ \2,\1 /g' file2.txt
Heddings,Lowell
Heinzman,Andrew
Hendrickson,Josh
Hoffman,Chris
Iwaya,Akemi
McKay,Dave
Summerson,Cameron
vvdn-user@vvdn-user:~/Documents/dir2$
```

To add text to the start of a line, we'll use a substitution command that matches everything on the line, combined with a replacement clause that combines our new text with the original line.



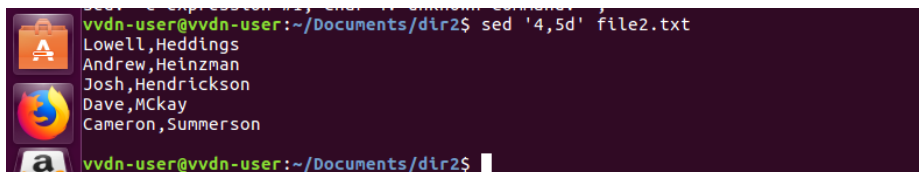
```
vvdn-user@vvdn-user:~/Documents/dir2$ sed 's/.*/--> INSERTED! &/' file2.txt
--> INSERTED! Lowell,Heddings
--> INSERTED! Andrew,Heinzman
--> INSERTED! Josh,Hendrickson
--> INSERTED! Chris,Hoffman
--> INSERTED! Akemi,Iwaya
--> INSERTED! Dave,McKay
--> INSERTED! Cameron,Summerson
--> INSERTED!
vvdn-user@vvdn-user:~/Documents/dir2$
```

Including the G command, which will add a blank line between each line:



```
vvdn-user@vvdn-user:~/Documents/dir2$ sed 'G' file2.txt
Lowell,Heddings
Andrew,Heinzman
Josh,Hendrickson
Chris,Hoffman
Akemi,Iwaya
Dave,Mckay
Cameron,Summerson
vvdn-user@vvdn-user:~/Documents/dir2$
```

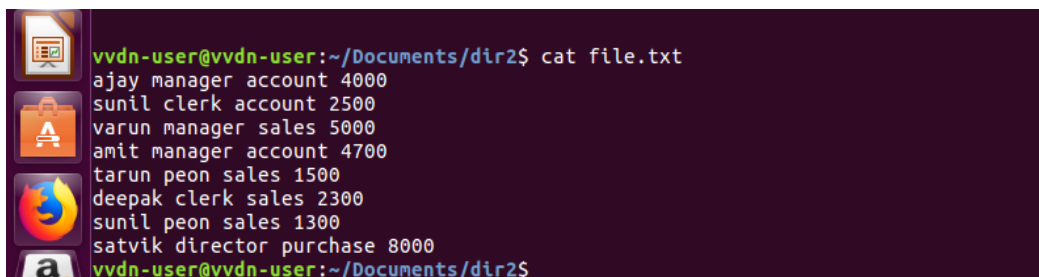
To delete the range of lines four to five,



```
vvdn-user@vvdn-user:~/Documents/dir2$ sed '4,5d' file2.txt
Lowell,Heddings
Andrew,Heinzman
Josh,Hendrickson
Dave,Mckay
Cameron,Summerson
vvdn-user@vvdn-user:~/Documents/dir2$
```

AWK COMMANDS:

Awk is a scripting language used for manipulating data and generating reports. The awk command programming language requires no compiling, and allows the user to use variables, numeric functions, string functions, and logical operators. Awk is mostly used for pattern scanning and processing. It searches one or more files to see if they contain lines that matches with the specified patterns and then performs the associated actions.



```
vvdn-user@vvdn-user:~/Documents/dir2$ cat file.txt
ajay manager account 4000
sunil clerk account 2500
varun manager sales 5000
amit manager account 4700
tarun peon sales 1500
deepak clerk sales 2300
sunil peon sales 1300
satvik director purchase 8000
vvdn-user@vvdn-user:~/Documents/dir2$
```

Default behavior of Awk :

By default Awk prints every line of data from the specified file.

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk '{print}' file.txt
ajay manager account 4000
sunil clerk account 2500
varun manager sales 5000
amit manager account 4700
tarun peon sales 1500
deepak clerk sales 2300
sunil peon sales 1300
satvik director purchase 8000
vvdn-user@vvdn-user:~/Documents/dir2$
```

To Print the lines which matches with the given pattern:

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk '/manager/{print}' file.txt
ajay manager account 4000
varun manager sales 5000
amit manager account 4700
vvdn-user@vvdn-user:~/Documents/dir2$
```

Splitting a Line Into Fields :

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk '{print $1,$4 }' file.txt
ajay 4000
sunil 2500
varun 5000
amit 4700
tarun 1500
deepak 2300
sunil 1300
satvik 8000
vvdn-user@vvdn-user:~/Documents/dir2$ awk '{print $1,$3 }' file.txt
ajay account
sunil account
varun sales
amit account
tarun sales
deepak sales
sunil sales
satvik purchase
vvdn-user@vvdn-user:~/Documents/dir2$
```

Awk's built-in variables include the field variables—\$1, \$2, \$3, and so on (\$0 is the entire line) — that break a line of text into individual words or pieces called fields.

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.

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk '{print NR,$0}' file.txt
1 ajay manager account 4000
2 sunil clerk account 2500
3 varun manager sales 5000
4 amit manager account 4700
5 tarun peon sales 1500
6 deepak clerk sales 2300
7 sunil peon sales 1300
8 satvik director purchase 8000
vvdn-user@vvdn-user:~/Documents/dir2$
```

Another use of NR built-in variables (Display Line From 3 to 6)

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk 'NR==3, NR==6 {print NR,$0}' file.txt
3 varun manager sales 5000
4 amit manager account 4700
5 tarun peon sales 1500
6 deepak clerk sales 2300
vvdn-user@vvdn-user:~/Documents/dir2$
```

NF: NF command keeps a count of the number of fields within the current input record.

```
Terminal
vvdn-user@vvdn-user: ~/Documents/dir2
vvdn-user@vvdn-user:~/Documents/dir2$ awk '{print $1,$NF}' file.txt
ajay 4000
sunil 2500
varun 5000
amit 4700
tarun 1500
deepak 2300
sunil 1300
satvik 8000
vvdn-user@vvdn-user:~/Documents/dir2$ awk '{print $2,$NF}' file.txt
manager 4000
clerk 2500
manager 5000
manager 4700
peon 1500
clerk 2300
peon 1300
director 8000
```

To count the lines in a file:

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk 'END {print NR}' file.txt
8
```

Printing lines with more than specified characters:

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk 'length($0) > 23' file.txt
ajay manager account 4000
sunil clerk account 2500
varun manager sales 5000
amit manager account 4700
satvik director purchase 8000
vvdn-user@vvdn-user:~/Documents/dir2$ awk 'length($0) > 25' file.txt
satvik director purchase 8000
vvdn-user@vvdn-user:~/Documents/dir2$
```

To print the first item along with the row number(NR) separated with " - " from each line:

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk '{print NR "-" $1}' file.txt
1-ajay
2-sunil
3-varun
4-amit
5-tarun
6-deepak
7-sunil
8-satvik
```

To return the third row/item :

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk '{print $3 }' file.txt
account
account
sales
account
sales
sales
sales
sales
purchase
vvdn-user@vvdn-user:~/Documents/dir2$
```

To print "Start Processing.", then print the third field of each record and finally "End Processing" :

```
rint $3}; END {print "end processing"}' file.txt
start processing.
account
account
sales
account
sales
sales
sales
sales
purchase
end processing
vvdn-user@vvdn-user:~/Documents/dir2$ awk 'BEGIN {print "start processing."}; {p
rint $4}; END {print "end processing"}' file.txt
start processing.
4000
2500
5000
4700
1500
2300
1300
8000
end processing
vvdn-user@vvdn-user:~/Documents/dir2$
```

using the awk command in shell scripts:

```
vvdn-user@vvdn-user:~/Documents/dir2$ num=50
vvdn-user@vvdn-user:~/Documents/dir2$ awk -v n="$num" 'BEGIN {print "the number
is",n}'
the number is 50
vvdn-user@vvdn-user:~/Documents/dir2$
```

To print the squares and cubes of first numbers:

```
vvdn-user@vvdn-user:~/Documents/dir2$ awk 'BEGIN {for(i=1;i<=5;i++) print "suar
e value of",i,"is",i*i;}'
square value of 1 is 1
square value of 2 is 4
square value of 3 is 9
square value of 4 is 16
square value of 5 is 25
vvdn-user@vvdn-user:~/Documents/dir2$ awk 'BEGIN {for(i=1;i<=6;i++) print "cube
value of",i,"is",i*i*i;}'
cube value of 1 is 1
cube value of 2 is 8
cube value of 3 is 27
cube value of 4 is 64
cube value of 5 is 125
cube value of 6 is 216
vvdn-user@vvdn-user:~/Documents/dir2$
```