1. **Text Editors Overview**
2. **Vi/Vim**
3. **Text Processors**
4. **Text Editors Overview**

There is a very wide selection of Linux text editors from which to choose for almost any distribution (flavor of Linux). Some will be in the default installation, others will only be seen in a particular desktop environment, while still others can only be obtained from the repository (a machine that holds Linux packages) for your specific distribution (CentOS in our case). Linux text editors vary greatly in features and some have a much steeper learning curve than others. It's a good idea to work with a few different ones until you settle on a suitable choice for all or most of the tasks you do.

You might have also heard of the more well known **emacs** and **vi** editors. Keep in mind that some Linux text editors are console-based (such as the vi editor) while others have a GUI. Well, in our case we will stick only to the text based editors.

The following are all Linux text editors:

**emacs** - This is much more than a text editor. Richard Stallman's emacs editor can be used as a calendar and appointment book, to send e-mail, play games, and more.

**mcedit** - People who like DOS-type editors will like working with mcedit. This editor is included in the package with mc binary.

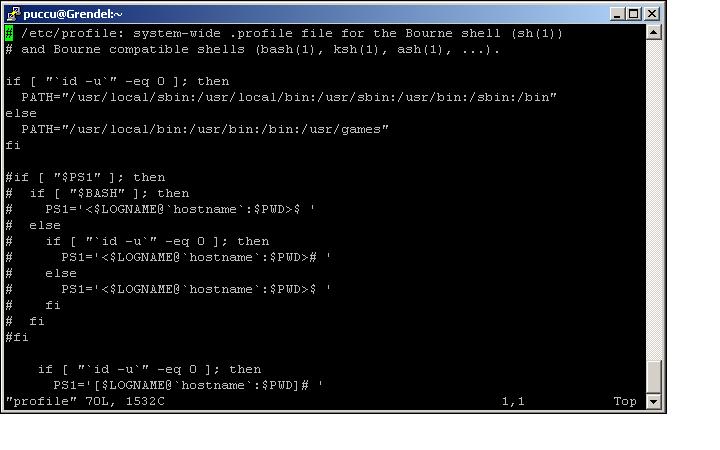
**joe** - Known as Joe's Own Editor, it can be used to emulate some other editors.

**vi** - This one was written by Bill Joy. There's also the **vim (Vi IMproved)** version available on practically every Linux system.

1. **Vi/Vim**

The advantage of learning **vi** and learning it well is that one will find **vi** on all Unix based systems and it does not consume an inordinate amount of system resources. Vi works great over slow network ppp modem connections and on systems of limited resources. One can completely utilize vi without departing a single finger from the keyboard. (No hand to mouse and return to keyboard latency).

Editing a file in vi is done by issuing the command: **vi file-to-edit.txt** (ex : **vi profile**)



The vi editor has three modes, command mode, insert mode and command line mode.

1. **Command mode:** letters or sequence of letters interactively command vi. Commands are case sensitive. The ESC key can end a command.
2. **Insert mode:** Text is inserted. The ESC key ends insert mode and returns you to command mode. From *Command mode* you can enter *Insert mode* with the "i" (insert), "a" (insert after), "A" (insert at end of line), "o" (open new line after current line) or "O" (Open line above current line) commands.
3. **Command line mode:** You enter this mode by typing ":" which puts the command line entry at the foot of the screen.

**Vi in Command Mode:**

|  |  |
| --- | --- |
| i | Switch to insert mode and type in your text |
| a | Append to current character position |
| A | Append to the end of the line |
| o | Jump to the beginning of a new below line |
| O | Jump to the beginning of a new above line |
| 0 | Jump to the beginning of the line |
| $ | Jump to the end of the line |
| 1 + G | Jump to the first line in a file |
| G | Jump to the last line of a file |
| 5 + G | Jump to the 5th line within a file |
| W | Jump a word to right |
| B | Jump a word to left |
| h | Move to left a character |
| l | Move to right a character |
| j | Move down a character |
| k | Move up a character |
| /key | Search key from up to down |
| ?key | Search key from down to up |
| yy | Copy the current line |
| 7 + yy | Copy 7 lines from the current posiont |
| p | Paste |
| u | Undo |
| Ctrl+r | Redo |
| dd | Delete current line |
| dw | Delete from current position to the end of word |
| db | Delete from current position to the beginning of the word |
| 3 + dd | Delete 3 lines |
| D | Delete from current position to the end of line |
| d$ | Delete from current position to the end of line |
| d0 | Delete from current position to the beginning of line |
| x | Delete one character at time |
| X | Delete everything before current position |
| r | Replace current character |
| R | Replace Mode -> ESC to return to Command Mode |
| ( | Move by sentence forward |
| ) | Move by sentence backword |
| { | Move by paragraph forward |
| } | Move by paragraph backword |

**Vi in Command Line Mode:**

|  |  |
| --- | --- |
| :q | Quite (if the file was not modified) |
| :wq | Write and quite |
| :w! | Force write |
| :q! | Quite without saving (if the file was modified) |
| :!command | Execute commands in shell and get output in vi |
| :r file | Reads from file |
| :w file | Writes in file |
| :split | Split the screen horizontally |
| :vsplit | Split the screen vertically |
| :close | Close a vi window |
| Ctrl+w | Move between vi windows |
| :sh | Escape to shell |
| :set nu | Set line numbering |
| :set nonu | Unset line numbering |
| :set ignorecase | Set vi to ignore case sensitive |
| :help | Get help |

1. **Text Processors**

Text processors are usefull commands for processing text that can be executed directly in the shell, or in a shell script. Here are some text processors within Linux environment:

**GREP** – GNU Regular Expression Print

Grep is considered to be the most used text processor and and it also has a variant for expressions/patterns, egrep. The syntax is:

***Grep search\_pattern filename …***

The command prints lines that contain the given search pattern. You can also specify several files from which grep will print thematching line. Here are some example:

**Grep “^#” file** - grep every line that starts with a #

**Grep –v “$#” file** - grep every line that doesn’t end with a #

**Grep –i test /etc/passwd** - grep with ignore case sensitive function (-i)

**Grep –n “^test[0-9]\*” file** - grep for every line that starts with test, contains some numbers between 0 to 9 and end with everything. Also displays the line number.

**SED** – Stream Editor

Sed is an editor that performs text transformations on a line-by-line basis. The syntax is:

***Sed editing-command filename***

Each sed command must be preceded by an exact address or address range specifying the lines to which the editing command applies. Here are some examples:

**Sed ‘s/cat/dog’’ file** - search the first line containing cat and replace with dog

**Sed ‘s/cat/dog/g’ file** - the same as the above but for all files (g - globally)

**Sed ‘1,50/cat/dog/g’ file** – the same as the above but only for the first 50 lines

**Sed –n ‘1,9p’ file** - display line 1 to 9 from file

**Sed ’10,$d’ file**  - display line 1 to 9 from file and delete everything from line 10

**CUT**

You can use the cut command to cut out sections of line from a file, so only the specified section is printed on standard output. Here are some cut options and examples:

-d : specifies the delimiter

-f : specifies the column to print

-c : cuts by characters

[root@Oma.lan:/root]# cut -d : -f1 /etc/passwd | tail -1

alex

[root@Oma.lan:/root]# cut -d : -f1 /etc/passwd | head -1

Root

In the above example we have used “:” as delimiter, printed only the first column of /etc/passwd, the one that displays the users and then redirected the output as input to the commands tail and head. Tail is used to display the last 10 lines in a file (by default) and head the first 10 (by default). Head and tail permit you to modify the number of displayed lines by using the - <number> as show in the above example.

**TR** – Translator

TR command translates (replaces) or deletes characters. It reads from standard input and prints the result on the standard output. The syntax is the following

***Tr set1 set2***

The characters included in set1 are replaced with the characters included in set2. Here are some examples:

**Cat file | tr a-z A-Z** - change all lowercase characters to uppercase and prints it on screen

**Cat file | tr –d “:”** - delete all “:” from file and print the result on screen

**Cat file | tr –s ‘:’ ‘+’** - searches and replaces ‘:’ with ‘+’

Notice that in the above examples the translated text file was NOT automatically saved on the original one. If you want this as well you must us tr directly on the file or use an overwrite redirection operation (>) to the original or a new file.

**AWK**

AWK command is used to scan the file line by line, break each line into columns, based on some column separator (by default space “ ” is used as separator) and assign a variable to each column.

$0 – represents all columns

$1 – represents the first column

…

Here are some examples:

[root@Oma.lan:/tmp]# **ps aux | grep -e "/sbin/apache" | awk '{print $0}'**

www-data 4560 0.0 0.5 46500 11000 ? S< Mar20 0:00 /usr/sbin/apache2 -k start

www-data 4561 0.0 0.2 27248 5368 ? S< Mar20 0:00 /usr/sbin/apache2 -k start

www-data 4563 0.0 0.5 45972 10452 ? S< Mar20 0:00 /usr/sbin/apache2 -k start

root 7123 0.0 0.4 27080 8608 ? S<s Feb27 0:31 /usr/sbin/apache2 -k start

root 16836 0.0 0.0 3120 708 pts/0 R<+ 20:41 0:00 grep -e /sbin/apache

www-data 29181 0.0 0.5 46240 10520 ? S< Mar26 0:00 /usr/sbin/apache2 -k start

www-data 32506 0.0 0.5 46268 10816 ? S< Mar21 0:01 /usr/sbin/apache2 -k start

www-data 32507 0.0 0.5 46252 10864 ? S< Mar21 0:00 /usr/sbin/apache2 -k start

www-data 32508 0.0 0.5 46504 11096 ? S< Mar21 0:02 /usr/sbin/apache2 -k start

www-data 32516 0.0 0.5 46224 10320 ? S< Mar21 0:00 /usr/sbin/apache2 -k start

www-data 32517 0.0 0.3 46000 7288 ? S< Mar21 0:00 /usr/sbin/apache2 -k start

www-data 32519 0.0 0.2 27504 5496 ? S< Mar21 0:00 /usr/sbin/apache2 -k start

[root@Oma.lan:/tmp]# **ps aux | grep -e "/sbin/apache" | awk '{print $2}'**

4560

4561

4563

7123

16840

29181

32506

32507

32508

32516

32517

32519

[root@Oma.lan:/tmp]# **cat passwd | awk -F : '{print $1, "--->", $7}' | head -4**

root ---> /bin/bash

daemon ---> /bin/sh

bin ---> /bin/sh

sys ---> /bin/sh

In the first example we want to display all apache web server processes and by using awk, we extracted all columns in $0 variable. In the second example by printing only $2 we gathered only the PIDs of the apache processes. In the third example we have changed the column separator from space to “:” using –F parameter for the filename “passwd” and then we printed the first column, which is represented by the username, and then the 7th column which is represent by the shell. Also we have displayed only the first 4 lines from the file.