Introduction to `Bash`

Text Editors, Process and Redirection

Mauricio Sevilla

`email= j.sevillam@uniandes.edu.co`

01.28.19

Review

We have already talk about the terminal and some of its uses.

What have we learned?

- *Differences between `GUI` and `CLI`*
- *Some instructions such as*
    - `more`
    - `less`
    - `cat`
    - `cd`
    - `cp`
    - `ls`
    - `wc`
    - `pwd`

Do you remember how to use them?

First, you should open `Binder` , a new link is available on our repository

> [*github.com/jmsevillam/Herramientas-Computacionales-UniAndes (https://github.com/jmsevillam/Herramientas-Computacionales-UniAndes)*](https://github.com/jmsevillam/Herramientas-Computacionales-UniAndes)

There you will find installed the packages we are going to use during the classes.

After that, open a `terminal` window.

Last class, we created a small file, by using the operator `>` ,

Question

What is the difference between `>` and `>>` .

let us explore it

Let us create a file by using `>` as we have already done,

In [1]:
```
echo this is a test > test.txt
```

We learned that now on the file there must be saved `this is a test`, try to view the content

In [2]:
```
cat test.txt
```

```
this is a test
```

But, what if we want to add a new line,

```
In [3]: echo this is a new test > test.txt
```

Again, we can see the file

```
In [4]: cat test.txt
```

```
this is a new test
```

The second instruction erased the first one, that wasn't what we wanted, let's do it different

Let's remove the file, just in order to be sure we are creating a new one

In [5]:
```
rm test.txt
```

Again, we can start in the same way we did before

In [6]:
```
echo this is a test > test.txt
```

we can open the file

In [7]:
```
cat test.txt
```

```
this is a test
```

if we use >> instead of >

In [8]:
```
echo this is a new test >> test.txt
```

and print the result

In [10]:
```
cat test.txt
```

```
this is a test
this is a new test
```

Conclusions

The use uf `>` and `>>` depends strongly on what we want to do.

> *`>` replaces the content of a file, if doesn't exist, creates it. `>>` Adds a new line of the file.*

This process is called Redirection.

Some times we may be interested on doing a bigger file, and using a line by line addition is not the best option, there are some programs which can help us to make this task easier.

Right now, there are may of these programs, some of the most used are

>

- *vi*
- *vim*
- *emacs*
- *nano*   *try them.*

```
In [19]: man vim
```

```
VIM(1)                                                                  VIM(1)



NAME
       vim - Vi IMproved, a programmer's text editor

SYNOPSIS
       vim [options] [file ..]
       vim [options] -
       vim [options] -t tag
       vim [options] -q [errorfile]

       ex
       view
       gvim gview evim eview
       rvim rview rgvim rgview

DESCRIPTION
       Vim  is a text editor that is upwards compatible to Vi.  It can be used
       to edit all kinds of plain text.  It is especially useful  for  editing
       programs.

       There  are a lot of enhancements above Vi: multi level undo, multi win-
       dows and buffers, syntax highlighting, command line  editing,  filename
       completion,   on-line   help,   visual   selection,   etc..   See  ":help
       vi_diff.txt" for a summary of the differences between Vim and Vi.

       While running Vim a lot of help can be obtained from the  on-line  help
       system,  with the ":help" command.  See the ON-LINE HELP section below.

       Most often Vim is started to edit a single file with the command

            vim file
```

More generally Vim is started with:

```
vim [options] [filelist]
```

If the filelist is missing, the editor will start with an empty buffer. Otherwise exactly one out of the following four may be used to choose one or more files to be edited.

```
file ..     A list of filenames.  The first one  will  be  the  current
            file  and  read  into the buffer.  The cursor will be posi-
            tioned on the first line of the buffer.  You can get to the
            other  files with the ":next" command.  To edit a file that
            starts with a dash, precede the filelist with "--".

-           The file to edit is read from  stdin.   Commands  are  read
            from stderr, which should be a tty.

-t {tag}    The file to edit and the initial cursor position depends on
            a "tag", a sort of goto label.  {tag} is looked up  in  the
            tags file, the associated file becomes the current file and
            the associated command is executed.  Mostly  this  is  used
            for  C  programs,  in  which case {tag} could be a function
            name.  The effect is that the file containing that function
            becomes  the  current  file and the cursor is positioned on
            the start of the function.  See ":help tag-commands".

-q [errorfile]
            Start in quickFix mode.  The file [errorfile] is  read  and
            the  first  error is displayed.  If [errorfile] is omitted,
            the  filename  is  obtained  from  the  'errorfile'  option
            (defaults  to  "AztecC.Err"  for the Amiga, "errors.err" on
            other systems).  Further errors can be jumped to  with  the
            ":cn" command.  See ":help quickfix".
```

Vim behaves differently, depending on the name of the command (the executable may still be the same file).

```
       vim       The "normal" way, everything is default.

       ex        Start in Ex mode.  Go to Normal mode with the ":vi"  command.
                 Can also be done with the "-e" argument.

       view      Start  in read-only mode.  You will be protected from writing
                 the files.  Can also be done with the "-R" argument.

       gvim gview
                 The GUI version.  Starts a new window.  Can also be done with
                 the "-g" argument.

       evim eview
                 The GUI version in easy mode.  Starts a new window.  Can also
                 be done with the "-y" argument.

       rvim rview rgvim rgview
                 Like the above, but with restrictions.  It will not be possi-
                 ble  to  start  shell  commands, or suspend Vim.  Can also be
                 done with the "-Z" argument.

OPTIONS
       The options may be given in  any  order,  before  or  after  filenames.
       Options without an argument can be combined after a single dash.

       +[num]      For  the  first  file the cursor will be positioned on line
                   "num".  If "num" is missing, the cursor will be  positioned
                   on the last line.

       +/{pat}     For  the  first  file  the cursor will be positioned in the
                   line with  the  first  occurrence  of  {pat}.   See  ":help
                   search-pattern" for the available search patterns.

       +{command}

       -c {command}
                   {command}  will  be  executed after the first file has been
                   read.  {command} is interpreted as an Ex command.   If  the
```

{command} contains spaces it must be enclosed in double quotes (this depends on the shell that is used). Example: Vim "+set si" main.c
Note: You can use up to 10 "+" or "-c" commands.

-S {file}   {file} will be sourced after the first file has been read. This is equivalent to -c "source {file}". {file} cannot start with '-'. If {file} is omitted "Session.vim" is used (only works when -S is the last argument).

--cmd {command}
Like using "-c", but the command is executed just before processing any vimrc file. You can use up to 10 of these commands, independently from "-c" commands.

-A   If Vim has been compiled with ARABIC support for editing right-to-left oriented files and Arabic keyboard mapping, this option starts Vim in Arabic mode, i.e. 'arabic' is set. Otherwise an error message is given and Vim aborts.

-b   Binary mode. A few options will be set that makes it possible to edit a binary or executable file.

-C   Compatible. Set the 'compatible' option. This will make Vim behave mostly like Vi, even though a .vimrc file exists.

-d   Start in diff mode. There should be two, three or four file name arguments. Vim will open all the files and show differences between them. Works like vimdiff(1).

-d {device} Open {device} for use as a terminal. Only on the Amiga. Example: "-d con:20/30/600/150".

-D   Debugging. Go to debugging mode when executing the first command from a script.

-e   Start Vim in Ex mode, just like the executable was called

```
                "ex".

       -E          Start Vim in improved Ex mode, just like the executable was
                   called "exim".

       -f          Foreground.  For the GUI version, Vim  will  not  fork  and
                   detach from the shell it was started in.  On the Amiga, Vim
                   is not restarted to open a new window.  This option  should
                   be  used  when  Vim is executed by a program that will wait
                   for the edit session to finish (e.g. mail).  On  the  Amiga
                   the ":sh" and ":!" commands will not work.

       --nofork    Foreground.   For  the  GUI  version, Vim will not fork and
                   detach from the shell it was started in.

       -F          If Vim has been compiled with  FKMAP  support  for  editing
                   right-to-left  oriented  files  and Farsi keyboard mapping,
                   this option starts Vim in  Farsi  mode,  i.e.  'fkmap'  and
                   'rightleft'  are  set.  Otherwise an error message is given
                   and Vim aborts.

       -g          If Vim has been compiled  with  GUI  support,  this  option
                   enables  the  GUI.   If  no GUI support was compiled in, an
                   error message is given and Vim aborts.

       -h          Give a bit of help about the  command  line  arguments  and
                   options.  After this Vim exits.

       -H          If Vim has been compiled with RIGHTLEFT support for editing
                   right-to-left oriented files and Hebrew  keyboard  mapping,
                   this  option  starts  Vim  in Hebrew mode, i.e. 'hkmap' and
                   'rightleft' are set.  Otherwise an error message  is  given
                   and Vim aborts.

       -i {viminfo}
                   When  using  the  viminfo file is enabled, this option sets
                   the filename to use, instead of the  default  "~/.viminfo".
                   This can also be used to skip the use of the .viminfo file,
               by giving the name "NONE".

       -L          Same as -r.

       -l          Lisp mode.  Sets the 'lisp' and 'showmatch' options on.

       -m          Modifying files is disabled.  Resets  the  'write'  option.
                   You  can still modify the buffer, but writing a file is not
                   possible.

       -M          Modifications not allowed.  The  'modifiable'  and  'write'
                   options  will be unset, so that changes are not allowed and
                   files can not be written.  Note that these options  can  be
                   set to enable making modifications.

       -N          No-compatible  mode.   Reset the 'compatible' option.  This
                   will make Vim behave a bit better, but less Vi  compatible,
                   even though a .vimrc file does not exist.

       -n          No  swap file will be used.  Recovery after a crash will be
                   impossible.  Handy if you want to edit a  file  on  a  very
                   slow  medium  (e.g.  floppy).   Can also be done with ":set
                   uc=0".  Can be undone with ":set uc=200".

       -nb         Become an editor server for NetBeans.   See  the  docs  for
                   details.

       -o[N]       Open N windows stacked.  When N is omitted, open one window
                   for each file.

       -O[N]       Open N windows side by side.  When N is omitted,  open  one
                   window for each file.

       -p[N]       Open N tab pages.  When N is omitted, open one tab page for
                   each file.

       -R          Read-only mode.  The 'readonly' option will  be  set.   You
                   can still edit the buffer, but will be prevented from acci-
                   dentally overwriting a file.  If you do want to overwrite a
                   file,  add  an  exclamation  mark  to the Ex command, as in
                   ":w!".  The -R option  also  implies  the  -n  option  (see
                   above).   The  'readonly'  option  can  be reset with ":set
                   noro".  See ":help 'readonly'".

       -r             List swap files, with  information  about  using  them  for
                   recovery.

       -r {file}      Recovery  mode.  The swap file is used to recover a crashed
                   editing session.  The swap file is a  file  with  the  same
                   filename as the text file with ".swp" appended.  See ":help
                   recovery".

       -s             Silent mode.  Only when started as "Ex" or  when  the  "-e"
                   option was given before the "-s" option.

       -s {scriptin}
                   The  script file {scriptin} is read.  The characters in the
                   file are interpreted as if you had typed  them.   The  same
                   can be done with the command ":source! {scriptin}".  If the
                   end of the file is reached before the editor exits, further
                   characters are read from the keyboard.

       -T {terminal}
                   Tells  Vim  the  name  of the terminal you are using.  Only
                   required when the automatic way doesn't work.  Should be  a
                   terminal  known  to Vim (builtin) or defined in the termcap
                   or terminfo file.

       -u {vimrc}     Use the commands in the file {vimrc}  for  initializations.
                   All  the  other  initializations  are skipped.  Use this to
                   edit a special kind of files.  It can also be used to  skip
                   all  initializations by giving the name "NONE".  See ":help
                   initialization" within vim for more details.

       -U {gvimrc}    Use the commands in the file {gvimrc} for  GUI  initializa-
                   tions.   All the other GUI initializations are skipped.  It
```

can also be used to skip all GUI initializations by giving the name "NONE". See ":help gui-init" within vim for more details.

-V[N] Verbose. Give messages about which files are sourced and for reading and writing a viminfo file. The optional number N is the value for 'verbose'. Default is 10.

-v Start Vim in Vi mode, just like the executable was called "vi". This only has effect when the executable is called "ex".

-w {scriptout}
All the characters that you type are recorded in the file {scriptout}, until you exit Vim. This is useful if you want to create a script file to be used with "vim -s" or ":source!". If the {scriptout} file exists, characters are appended.

-W {scriptout}
Like -w, but an existing file is overwritten.

-x Use encryption when writing files. Will prompt for a crypt key.

-X Don't connect to the X server. Shortens startup time in a terminal, but the window title and clipboard will not be used.

-y Start Vim in easy mode, just like the executable was called "evim" or "eview". Makes Vim behave like a click-and-type editor.

-Z Restricted mode. Works like the executable starts with "r".

-- Denotes the end of the options. Arguments after this will be handled as a file name. This can be used to edit a

```
                 filename that starts with a '-'.

      --echo-wid  GTK GUI only: Echo the Window ID on stdout.

      --help      Give a help message and exit, just like "-h".

      --literal   Take file name arguments literally,  do  not  expand  wild-
                  cards.   This has no effect on Unix where the shell expands
                  wildcards.

      --noplugin  Skip loading plugins.  Implied by -u NONE.

      --remote    Connect to a Vim server and make it edit the files given in
                  the rest of the arguments.  If no server is found a warning
                  is given and the files are edited in the current Vim.

      --remote-expr {expr}
                  Connect to a Vim server, evaluate {expr} in  it  and  print
                  the result on stdout.

      --remote-send {keys}
                  Connect to a Vim server and send {keys} to it.

      --remote-silent
                  As  --remote,  but  without  the  warning when no server is
                  found.

      --remote-wait
                  As --remote, but Vim does not exit  until  the  files  have
                  been edited.

      --remote-wait-silent
                  As --remote-wait, but without the warning when no server is
                  found.

      --serverlist
                  List the names of all Vim servers that can be found.
```

```
       --servername {name}
                   Use {name} as the server name.  Used for the  current  Vim,
                   unless used with a --remote argument, then it's the name of
                   the server to connect to.

       --socketid {id}
                   GTK GUI only: Use the GtkPlug  mechanism  to  run  gvim  in
                   another window.

       --version   Print version information and exit.

ON-LINE HELP
       Type  ":help"  in Vim to get started.  Type ":help subject" to get help
       on a specific subject.  For example: ":help ZZ" to  get  help  for  the
       "ZZ"  command.   Use <Tab> and CTRL-D to complete subjects (":help cmd-
       line-completion").  Tags are present to jump from one place to  another
       (sort of hypertext links, see ":help").  All documentation files can be
       viewed in this way, for example ":help syntax.txt".

FILES
       /usr/local/lib/vim/doc/*.txt
                      The Vim documentation files.  Use ":help  doc-file-list"
                      to get the complete list.

       /usr/local/lib/vim/doc/tags
                      The  tags file used for finding information in the docu-
                      mentation files.

       /usr/local/lib/vim/syntax/syntax.vim
                      System wide syntax initializations.

       /usr/local/lib/vim/syntax/*.vim
                      Syntax files for various languages.

       /usr/local/lib/vim/vimrc
                      System wide Vim initializations.

       ~/.vimrc       Your personal Vim initializations.
       /usr/local/lib/vim/gvimrc
                      System wide gvim initializations.

       ~/.gvimrc      Your personal gvim initializations.

       /usr/local/lib/vim/optwin.vim
                      Script used for the ":options" command, a  nice  way  to
                      view and set options.

       /usr/local/lib/vim/menu.vim
                      System wide menu initializations for gvim.

       /usr/local/lib/vim/bugreport.vim
                      Script to generate a bug report.  See ":help bugs".

       /usr/local/lib/vim/filetype.vim
                      Script  to  detect  the type of a file by its name.  See
                      ":help 'filetype'".

       /usr/local/lib/vim/scripts.vim
                      Script to detect the type of a  file  by  its  contents.
                      See ":help 'filetype'".

       /usr/local/lib/vim/print/*.ps
                      Files used for PostScript printing.

       For recent info read the VIM home page:
       <URL:http://www.vim.org/>

SEE ALSO
       vimtutor(1)

AUTHOR
       Most of Vim was made by Bram Moolenaar, with a lot of help from others.
       See ":help credits" in Vim.
       Vim is based on Stevie, worked on by: Tim Thompson,  Tony  Andrews  and
       G.R.  (Fred) Walter.  Although hardly any of the original code remains.
```

BUGS

Probably. See ":help todo" for a list of known problems.

Note that a number of things that may be regarded as bugs by some, are in fact caused by a too-faithful reproduction of Vi's behaviour. And if you think other things are bugs "because Vi does it differently", you should take a closer look at the vi_diff.txt file (or type :help vi_diff.txt when in Vim). Also have a look at the 'compatible' and 'cpoptions' options.

```
In [15]: man emacs
```

```
EMACS(1)                                                            EMACS(1)



NAME
       emacs - GNU project Emacs

SYNOPSIS
       emacs [ command-line switches ] [ files ...  ]

DESCRIPTION
       GNU  Emacs is a version of Emacs, written by the author of the original
       (PDP-10) Emacs, Richard Stallman.
       The primary documentation of GNU Emacs is  in  the  GNU  Emacs  Manual,
       which  you  can  read  using Info, either from Emacs or as a standalone
       program.  Please look there for complete and up-to-date  documentation.
       This  man  page  is  updated only when someone volunteers to do so; the
       Emacs maintainers' priority goal is to minimize the amount of time this
       man page takes away from other more useful projects.
       The  user functionality of GNU Emacs encompasses everything other Emacs
       editors do, and it is easily extensible since its editing commands  are
       written in Lisp.

       Emacs  has  an  extensive  interactive  help facility, but the facility
       assumes that you know how to  manipulate  Emacs  windows  and  buffers.
       CTRL-h or F1 enters the Help facility.  Help Tutorial (CTRL-h t) starts
       an interactive tutorial which can teach beginners the  fundamentals  of
       Emacs  in a few minutes.  Help Apropos (CTRL-h a) helps you find a com-
       mand given its functionality, Help Character  (CTRL-h  c)  describes  a
       given  character's  effect,  and  Help  Function (CTRL-h f) describes a
       given Lisp function specified by name.

       Emacs's Undo can undo several steps of modification to your buffers, so
       it is easy to recover from editing mistakes.
```

GNU Emacs's many special packages handle mail reading (RMail) and sending (Mail), outline editing (Outline), compiling (Compile), running subshells within Emacs windows (Shell), running a Lisp read-eval-print loop (Lisp-Interaction-Mode), automated psychotherapy (Doctor), and much more.

There is an extensive reference manual, but users of other Emacses should have little trouble adapting even without a copy. Users new to Emacs will be able to use basic features fairly rapidly by studying the tutorial and using the self-documentation features.

Emacs Options

The following options are of general interest:

```
file    Edit file.

+number Go to the line specified by  number  (do  not  insert  a  space
        between the "+" sign and the number).  This applies only to the
        next file specified.

+line:column
        Go to the specified line and column

-q      Do not load an init file.

-no-site-file
        Do not load the site-wide startup file.

-debug-init
        Enable Emacs Lisp debugger during the processing  of  the  user
        init  file  ~/.emacs.  This is useful for debugging problems in
        the init file.

-u user Load user's init file.

-t file Use specified file as the terminal instead of using  stdin/std-
```

```
        out.   This must be the first argument specified in the command
        line.

-version
        Display Emacs version information and exit.

The following options are lisp-oriented (these options are processed in
the order encountered):

-f function
        Execute the lisp function function.

-l file Load the lisp code in the file file.

-eval expr
        Evaluate the Lisp expression expr.

The following options are useful when running Emacs as a batch editor:

-batch  Edit  in  batch mode.  The editor will send messages to stderr.
        This option must be the first in the argument list.   You  must
        use -l and -f options to specify files to execute and functions
        to call.

-kill   Exit Emacs while in batch mode.

-L directory
        Add directory to the list of  directories  Emacs  searches  for
        Lisp files.

Using Emacs with X

Emacs  has been tailored to work well with the X window system.  If you
run Emacs from under X windows, it will create its own X window to dis-
play  in.   You  will probably want to start the editor as a background
process so that you can continue using your original window.

Emacs can be started with the following X switches:
```

```
-name name
        Specifies the name which should  be  assigned  to  the  initial
        Emacs  window.  This controls looking up X resources as well as
        the window title.

-title name
        Specifies the title for the initial X window.

-r      Display the Emacs window in reverse video.

-font font, -fn font
        Set the Emacs window's font to that  specified  by  font.   You
        will  find the various X fonts in the /usr/lib/X11/fonts direc-
        tory.  Note that Emacs will  only  accept  fixed  width  fonts.
        Under  the X11 Release 4 font-naming conventions, any font with
        the value "m" or "c" in the eleventh field of the font name  is
        a  fixed  width font.  Furthermore, fonts whose name are of the
        form widthxheight are generally fixed width,  as  is  the  font
        fixed.  See xlsfonts(1) for more information.

        When  you  specify  a  font, be sure to put a space between the
        switch and the font name.

-bw pixels
        Set the Emacs window's border width to  the  number  of  pixels
        specified by pixels.  Defaults to one pixel on each side of the
        window.

-ib pixels
        Set the window's internal border width to the number of  pixels
        specified  by pixels.  Defaults to one pixel of padding on each
        side of the window.


--geometry geometry
        Set the Emacs window's width, height, and  position  as  speci-
        fied.   The geometry specification is in the standard X format;
        see X(1) for more information.  The width and height are speci-
        fied  in  characters;  the  default is 80 by 24.  See the Emacs
        manual, section "Options for Window  Size  and  Position",  for
        information on how window sizes interact with selecting or des-
        electing the tool bar and menu bar.


-fg color
        On color displays, sets the color of the text.

        Use the command M-x list-colors-display for  a  list  of  valid
        color names.

-bg color
        On color displays, sets the color of the window's background.

-bd color
        On color displays, sets the color of the window's border.

-cr color
        On  color displays, sets the color of the window's text cursor.

-ms color
        On color displays, sets the color of the window's mouse cursor.

-d displayname, -display displayname
        Create  the  Emacs  window on the display specified by display-
        name.  Must be the first option specified in the command  line.

-nw     Tells  Emacs not to use its special interface to X.  If you use
        this switch when invoking Emacs from an xterm(1)  window,  dis-
        play is done in that window.

You can set X default values for your Emacs windows in your .Xresources
file (see xrdb(1)).  Use the following format:

       emacs.keyword:value
```

where value specifies the default value of keyword. Emacs lets you set default values for the following keywords:

font (class Font)
        Sets the window's text font.

reverseVideo (class ReverseVideo)
        If reverseVideo's value is set to on, the window will be displayed in reverse video.

bitmapIcon (class BitmapIcon)
        If bitmapIcon's value is set to on, the window will iconify into the "kitchen sink."

borderWidth (class BorderWidth)
        Sets the window's border width in pixels.

internalBorder (class BorderWidth)
        Sets the window's internal border width in pixels.

foreground (class Foreground)
        For color displays, sets the window's text color.

background (class Background)
        For color displays, sets the window's background color.

borderColor (class BorderColor)
        For color displays, sets the color of the window's border.

cursorColor (class Foreground)
        For color displays, sets the color of the window's text cursor.

pointerColor (class Foreground)
        For color displays, sets the color of the window's mouse cursor.

geometry (class Geometry)
        Sets the geometry of the Emacs window (as described above).

title (class Title)
Sets the title of the Emacs window.

iconName (class Title)
Sets the icon name for the Emacs window icon.

If you try to set color values while using a black and white display, the window's characteristics will default as follows: the foreground color will be set to black, the background color will be set to white, the border color will be set to grey, and the text and mouse cursors will be set to black.

Using the Mouse

The following lists the mouse button bindings for the Emacs window under X11.

MOUSE BUTTON	FUNCTION
left	Set point.
middle	Paste text.
right	Cut text into X cut buffer.
SHIFT-middle	Cut text into X cut buffer.
SHIFT-right	Paste text.
CTRL-middle	Cut text into X cut buffer and kill it.
CTRL-right	Select this window, then split it into two window s. Same as typing CTRL-x 2.
CTRL-SHIFT-left	X buffer menu -- hold the buttons and keys down, wa it for menu to appear, select buffer, and release. Mo ve mouse out of menu and release to cancel.
CTRL-SHIFT-middle	X help menu -- pop up index card menu for Emacs hel p.
CTRL-SHIFT-right	Select window with mouse, and delete all other wi n-

dows. Same as typing CTRL-x 1.

MANUALS

You can order printed copies of the GNU Emacs Manual from the Free Software Foundation, which develops GNU software. See the file ORDERS for ordering information.
Your local Emacs maintainer might also have copies available. As with all software and publications from FSF, everyone is permitted to make and distribute copies of the Emacs manual. The TeX source to the manual is also included in the Emacs source distribution.

FILES

/usr/local/share/info - files for the Info documentation browser. The complete text of the Emacs reference manual is included in a convenient tree structured form. Also includes the Emacs Lisp Reference Manual, useful to anyone wishing to write programs in the Emacs Lisp extension language.

/usr/local/share/emacs/$VERSION/lisp - Lisp source files and compiled files that define most editing commands. Some are preloaded; others are autoloaded from this directory when used.

/usr/local/libexec/emacs/$VERSION/$ARCH - various programs that are used with GNU Emacs.

/usr/local/share/emacs/$VERSION/etc - various files of information.

/usr/local/share/emacs/$VERSION/etc/DOC.* - contains the documentation strings for the Lisp primitives and preloaded Lisp functions of GNU Emacs. They are stored here to reduce the size of Emacs proper.

/usr/local/share/emacs/$VERSION/etc/SERVICE lists people offering various services to assist users of GNU Emacs, including education, troubleshooting, porting and customization.

BUGS

There is a mailing list, bug-gnu-emacs@gnu.org, for reporting Emacs bugs and fixes. But before reporting something as a bug, please try to be sure that it really is a bug, not a misunderstanding or a deliberate feature. We ask you to read the section ``Reporting Emacs Bugs'' near the end of the reference manual (or Info system) for hints on how and when to report bugs. Also, include the version number of the Emacs you are running in every bug report that you send in.

Do not expect a personal answer to a bug report. The purpose of reporting bugs is to get them fixed for everyone in the next release, if possible. For personal assistance, look in the SERVICE file (see above) for a list of people who offer it.

Please do not send anything but bug reports to this mailing list. For more information about Emacs mailing lists, see the file /usr/local/emacs/etc/MAILINGLISTS. Bugs tend actually to be fixed if they can be isolated, so it is in your interest to report them in such a way that they can be easily reproduced.

UNRESTRICTIONS

Emacs is free; anyone may redistribute copies of Emacs to anyone under the terms stated in the Emacs General Public License, a copy of which accompanies each copy of Emacs and which also appears in the reference manual.

Copies of Emacs may sometimes be received packaged with distributions of Unix systems, but it is never included in the scope of any license covering those systems. Such inclusion violates the terms on which distribution is permitted. In fact, the primary purpose of the General Public License is to prohibit anyone from attaching any other restrictions to redistribution of Emacs.

Richard Stallman encourages you to improve and extend Emacs, and urges that you contribute your extensions to the GNU library. Eventually GNU (Gnu's Not Unix) will be a complete replacement for Unix. Everyone will be free to use, copy, study and change the GNU system.

SEE ALSO

emacsclient(1), etags(1), X(1), xlsfonts(1), xterm(1), xrdb(1)

AUTHORS

Emacs was written by Richard Stallman and the Free Software Foundation. Joachim Martillo and Robert Krawitz added the X features.

COPYING

In [16]:
```
man nano
```

```
NANO(1)                                                               NANO(1)



NAME
       nano - Nano's ANOther editor, an enhanced free Pico clone


SYNOPSIS
       nano [options] [[+line[,column]] file]...


DESCRIPTION
       nano  is  a  small and friendly editor.  It copies the look and feel of
       Pico, but is free software, and implements several features  that  Pico
       lacks,  such as: opening multiple files, scrolling per line, undo/redo,
       syntax coloring, line numbering, and soft-wrapping overlong lines.

       When giving a filename on the command line, the cursor can be put on  a
       specific line by adding the line number with a plus sign (+) before the
       filename, and even in a specific column by adding it with a comma.

       As a special case: if instead of a filename a dash (-) is  given,  nano
       will read data from standard input.


EDITING
       Entering  text  and  moving around in a file is straightforward: typing
       the letters and using the normal cursor movement  keys.   Commands  are
       entered by using the Control (^) and the Alt or Meta (M-) keys.  Typing
       ^K deletes the current line and puts it in the cutbuffer.   Consecutive
       ^Ks  will  put all deleted lines together in the cutbuffer.  Any cursor
       movement or executing any other command will cause the next ^K to over-
       write  the cutbuffer.  A ^U will paste the current contents of the cut-
```

buffer at the current cursor position.

When a more precise piece of text needs to be cut or copied, one can mark its start with ^6, move the cursor to its end (the marked text will be highlighted), and then use ^K to cut it, or M-6 to copy it to the cutbuffer. One can also save the marked text to a file with ^O, or spell check it with ^T.

On some terminals, text can be selected also by holding down Shift while using the arrow keys. Holding down the Ctrl or Alt key too will increase the stride. Any cursor movement without Shift being held will cancel such a selection.

The two lines at the bottom of the screen show some important commands; the built-in help (^G) lists all the available ones. The default key bindings can be changed via a nanorc file -- see nanorc(5).

OPTIONS

-A, --smarthome
: Make the Home key smarter. When Home is pressed anywhere but at the very beginning of non-whitespace characters on a line, the cursor will jump to that beginning (either forwards or backwards). If the cursor is already at that position, it will jump to the true beginning of the line.

-B, --backup
: When saving a file, back up the previous version of it, using the current filename suffixed with a tilde (~).

-C directory, --backupdir=directory
: Make and keep not just one backup file, but make and keep a uniquely numbered one every time a file is saved -- when backups are enabled (-B). The uniquely numbered files are stored in the specified directory.

-D, --boldtext
: Use bold text instead of reverse video text.

```
       -E, --tabstospaces
              Convert typed tabs to spaces.

       -F, --multibuffer
              Read a file into a new buffer by default.

       -G, --locking
              Use vim-style file locking when editing files.

       -H, --historylog
              Save the last hundred search strings and replacement strings and
              executed commands, so they can be easily reused  in  later  ses-
              sions.

       -I, --ignorercfiles
              Don't look at the system's nanorc nor at the user's nanorc.

       -K, --rebindkeypad
              Interpret  the  numeric  keypad keys so that they all work prop-
              erly.  You should only need to use this option if they don't, as
              mouse support won't work properly with this option enabled.

       -L, --nonewlines
              Don't  automatically add a newline when a file does not end with
              one.

       -M, --trimblanks
              Snip trailing whitespace from the wrapped  line  when  automatic
              hard-wrapping occurs or when text is justified.

       -N, --noconvert
              Disable automatic conversion of files from DOS/Mac format.

       -O, --morespace
              Use the blank line below the title bar as extra editing space.

       -P, --positionlog
```

For the 200 most recent files, log the last position of the cursor, and place it at that position again upon reopening such a file.

-Q "regex", --quotestr="regex"

Set the regular expression for matching the quoting part of a line. This is used when justifying. The default value is "^([\t]*([#:>|}]|//))+". Note that \t stands for an actual Tab.

-R, --restricted

Restricted mode: don't read or write to any file not specified on the command line. This means: don't read or write history files; don't allow suspending; don't allow spell checking; don't allow a file to be appended to, prepended to, or saved under a different name if it already has one; and don't make backup files. Restricted mode can also be activated by invoking nano with any name beginning with 'r' (e.g. "rnano").

-S, --smooth

Use smooth scrolling: text will scroll line-by-line, instead of the usual chunk-by-chunk behavior.

-T number, --tabsize=number

Set the size (width) of a tab to number columns. The value of number must be greater than 0. The default value is 8.

-U, --quickblank

Do quick status-bar blanking: status-bar messages will disappear after 1 keystroke instead of 25. Note that option -c (--constantshow) overrides this.

-V, --version

Show the current version number and exit.

-W, --wordbounds

Detect word boundaries differently by treating punctuation characters as part of a word.

```
-X "characters", --wordchars="characters"
       Specify  which other characters (besides the normal alphanumeric
       ones) should be considered as part of a  word.   This  overrides
       option -W (--wordbounds).

-Y name, --syntax=name
       Specify  the  name  of the syntax highlighting to use from among
       the ones defined in the nanorc files.

-Z, --zap
       Let an unmodified Backspace or Delete erase  the  marked  region
       (instead  of  a single character, and without affecting the cut-
       buffer).

-a, --atblanks
       When doing soft line wrapping, wrap lines at whitespace  instead
       of always at the edge of the screen.

-c, --constantshow
       Constantly  show  the  cursor  position on the status bar.  Note
       that this overrides option -U (--quickblank).

-d, --rebinddelete
       Interpret the Delete key differently so that both Backspace  and
       Delete  work  properly.  You should only need to use this option
       if Backspace acts like Delete on your system.

-g, --showcursor
       Make the cursor visible in the file browser (putting it  on  the
       highlighted  item)  and  in the help viewer.  Useful for braille
       users and people with poor vision.

-h, --help
       Show a summary of the available command-line options and exit.

-i, --autoindent
       Automatically indent a newly created line to the same number  of
```

```
              tabs  and/or spaces as the previous line (or as the next line if
              the previous line is the beginning of a paragraph).

       -k, --cutfromcursor
              Make the 'Cut Text' command (normally ^K) cut from  the  current
              cursor  position  to the end of the line, instead of cutting the
              entire line.

       -l, --linenumbers
              Display line numbers to the left of the text area.

       -m, --mouse
              Enable mouse   support,  if  available  for  your  system.   When
              enabled,  mouse  clicks can be used to place the cursor, set the
              mark (with a double click), and execute  shortcuts.   The  mouse
              will work in the X Window System, and on the console when gpm is
              running.  Text can still be selected through dragging by holding
              down the Shift key.

       -n, --noread
              Treat  any  name  given on the command line as a new file.  This
              allows nano to write to named pipes: it will start with a  blank
              buffer,  and  will  write  to  the  pipe when the user saves the
              "file".  This way nano can be used as an editor  in  combination
              with  for instance gpg without having to write sensitive data to
              disk first.

       -o directory, --operatingdir=directory
              Set the operating directory.  This makes nano set  up  something
              similar to a chroot.

       -p, --preserve
              Preserve  the XON and XOFF sequences (^Q and ^S) so they will be
              caught by the terminal.

       -q, --quiet
              Obsolete option.  Recognized but ignored.
```

```
-r number, --fill=number
       Hard-wrap lines at column number.  If this value is 0  or  less,
       wrapping  will occur at the width of the screen less number col-
       umns, allowing the wrap point to vary along with  the  width  of
       the  screen  if the screen is resized.  The default value is -8.
       This option conflicts with -w (--nowrap) -- the last  one  given
       takes effect.

-s program, --speller=program
       Use this alternative spell checker command.

-t, --tempfile
       Save  a changed buffer without prompting (when exiting with ^X).

-u, --unix
       Save a file by default in Unix format.   This  overrides  nano's
       default  behavior  of  saving  a file in the format that it had.
       (This option has no effect when you also use --noconvert.)

-v, --view
       Just view the file and disallow editing: read-only  mode.   This
       mode  allows  the  user  to  open  also other files for viewing,
       unless --restricted is given too.

-w, --nowrap
       Disable the hard-wrapping of long lines.  This option  conflicts
       with -r (--fill) -- the last one given takes effect.

-x, --nohelp
       Don't show the two help lines at the bottom of the screen.

-y, --afterends
       Make Ctrl+Right stop at word ends instead of beginnings.

-z, --suspend
       Enable the suspend ability.

-$, --softwrap
```

Enable 'soft wrapping'. This will make nano attempt to display the entire contents of any line, even if it is longer than the screen width, by continuing it over multiple screen lines. Since '$' normally refers to a variable in the Unix shell, you should specify this option last when using other options (e.g. 'nano -wS$') or pass it separately (e.g. 'nano -wS -$').

-b, -e, -f, -j
Ignored, for compatibility with Pico.

TOGGLES

Several of the above options can be switched on and off also while nano is running. For example, M-L toggles the hard-wrapping of long lines, M-$ toggles soft-wrapping, M-# toggles line numbers, M-M toggles the mouse, M-I auto-indentation, and M-X the help lines. See at the end of the ^G help text for a complete list.

INITIALIZATION FILE

nano will read two configuration files: first the system's nanorc (if it exists), and then the user's nanorc (if it exists), either ~/.nanorc or $XDG_CONFIG_HOME/nano/nanorc or ~/.config/nano/nanorc, whichever is encountered first. See nanorc(5) for more information on the possible contents of those files.

NOTES

If no alternative spell checker command is specified on the command line nor in one of the nanorc files, nano will check the SPELL environment variable for one.

In some cases nano will try to dump the buffer into an emergency file. This will happen mainly if nano receives a SIGHUP or SIGTERM or runs out of memory. It will write the buffer into a file named nano.save if the buffer didn't have a name already, or will add a ".save" suffix to the current filename. If an emergency file with that name already exists in the current directory, it will add ".save" plus a number (e.g. ".save.1") to the current filename in order to make it unique. In multibuffer mode, nano will write all the open buffers to their respective emergency files.

BUGS

Justifications (^J) are not yet covered by the general undo system. So after a justification that is not immediately undone, earlier edits cannot be undone any more. The workaround is, of course, to exit without saving.

The recording and playback of keyboard macros works correctly only on a terminal emulator, not on a Linux console (VT), because the latter does not by default distinguish modified from unmodified arrow keys.

Please report any other bugs that you encounter via:
https://savannah.gnu.org/bugs/?group=nano.

When nano crashes, it will save any modified buffers to emergency .save files. If you are able to reproduce the crash and you want to get a backtrace, define the environment variable NANO_NOCATCH.

HOMEPAGE

https://nano-editor.org/

SEE ALSO

nanorc(5)

/usr/share/doc/nano/ (or equivalent on your system)

AUTHOR

Chris Allegretta and others (see the files AUTHORS and THANKS for details). This manual page was originally written by Jordi Mallach for the Debian system (but may be used by others).

Exercise

Let's create some data using `echo` or any of the editors we have seen today.

For instance, a four word file which the following information,

Name	Code	Age	Career

Let's try to send this information to a file `usr.txt` were you must use your Andes email user instead of `usr`

Note Use spaces as separators!!

How can we download this data form `Binder`?

Send the file to `j.sevillam@uniandes.edu.co`.

(This will be counted as the assistence to the class for today.)

Why to do this?

This is done for you to see an example where `python` can be extremelly usefull (We are very close to start `python`)

Let see just a few examples of its use as a motivation before working directly with it.

Next class

We are going to keep exploring the shell, but a little bit more complex, we'll see commands such as

- *grep*
- *awk*

And we will learn how to write some basic scripts.

Next, we will start talking about `python`

Remember

For the next class, you should have seen the videos 1 and 2!!

During the day, the first assignment will be uploaded. (You will recieve it via `Email`)