UNIX Editors

Table of Contents

UNIX Editors - Introduction

- There are a multitude of **editors** under UNIX**:**
  - **Stream oriented** (`sed`, only for batch processing)
  - **Line oriented** (`ed`, `ex`)
  - **Screen oriented** (`vi`(`m`), `joe`, (`x`)`emacs`, `jed`, `pico`)
  - **Graphical oriented** (`kedit`, `textedit`, `nedit`)
- These so called **ASCII editors** are intended for the creation of configuration files, programs, mails, ...
- They **don‘t support text formatting** like **bold**, *italic*, or underlined.

UNIX Editors - Introduction

- From the lot of editors wie discuss the `vi`, because:
  - It is the **standard editor** under UNIX (*available in any case*).
  - It works **very fast** and may be **started fast**.
  - It needs only a small amount of **resources** (memory, network bandwidth).
  - Sometime is the only available editor (**rescue system**).
  - A lot of `vi` commands appear again **in other parts** of the UNIX system (`sed`, `awk`, `more`, ...).
  - Some commands start it **automatically** (`cron`, `visudo`, `vipasswd`, `v` in `more`/`less`).

UNIX Editors - Introduction

There is **no pressure** to use the vi!
Please use the editor
you like best.

But it pays a lot
to get aquainted with vi.

UNIX Editors - vi

Table of Contents (1)

vi - Introduction

- You may have **to get used to** the `vi` as it has a somewhat unusual logic of operation:
  - **Doesn‘t know** neither mouse nor cursor or function keys.
  - Is **modus oriented** because all functions are called via the normal typewriter keys.
  - Knows a lot of **movement commands**.
  - **First you have to key in the operation**, then you select the text to be operated on.
    - Graphical oriented systems do it the other way round: They **first mark the text** and then do some operations on it.

vi - Introduction

- Nearly each key (lower case, upper case, with `Ctrl`) triggers a command in the `vi`.
  - So it helps to be able **to key in with 10 fingers**.
  - **CAPS-LOCK** causes very unpleasent `vi` behaviour!
- You may memorize the meaning of each **command letter** easily by the (well selected) corresponding **English word**, e.g.:

`c`=change	`d`=delete	`s`=substitute
`a`=append	`i`=insert	`r`=replace

vi - Introduction

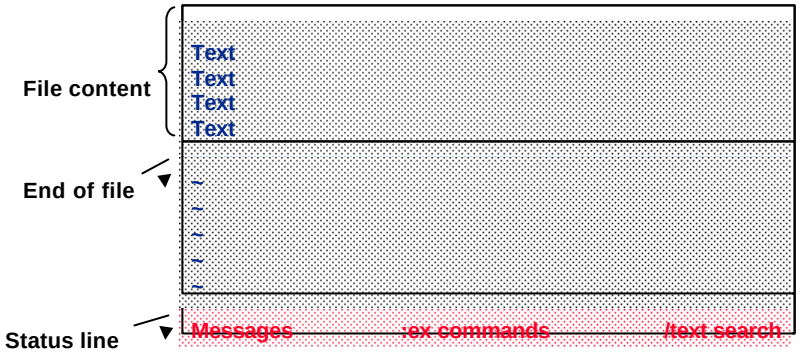
- Hidden behind the `vi` is its **line oriented** predecessor `ed` (**editor**) resp. `ex` (**editor extended**).
  - It may be activated temporarily for a command or even permanently.
- Under LINUX an extension of the `vi` called `vim` (**vi improved**) is used.
  - It is a real improvment of the `vi`.
  - There are a lot of other `vi` „**clones**“ (e.g. `vile`, `elvis`, `nvi`)

vi - Introduction

- The line at the bottom of the screen is a **status line**:
  - It is used to show warnings and error messages (*remain on the screen for some time*).
  - During `ex` **commands** or **text search** the cursor jumps into the status line too.
- Text **after the end of the file** will be marked by the character „~“ in the first column.
  - An empty file is marded by nothing but „~“ lines.

vi - Introduction

Look of Vi Screen ???



vi - Modi

- Because each letter is used as a command at the same time, the `vi` is **modus oriented**.
  - Dependent on the **current mode** pressing a key therefore has different results.
  - During working on a text one is **switching between the modes** back and forth all the time:

Command mode	Calling commands
Edit mode	Keying in text
Ex mode	Line oriented commands
Search mode	Searching text
(**Open mode**	Permanent ex mode)

vi - Modi

- **Central mode** is the **command mode**, which is active after `vi` has been started.
- To **switch** to another mode, the following input has to be given:

`a A c C i I o O r R s S`	→ **Edit mode**
`:`	→ **Ex mode**
`/ ?`	→ **Search mode**
`Q`	→ **Open mode**

(*Open mode is **never used**, but may be activated easily by mistake by the key „*`Q`*“*)

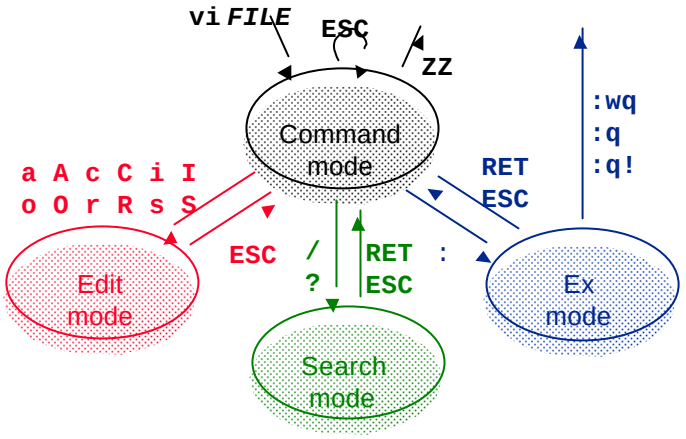
vi - Modi

- To **return** from the other modes back to the **command mode**, the following input has to be given:

ESC	→	Stops edit mode
↵	→	Executes command in ex / search mode
ESC	→	Stops ex / search mode
vi ↵	→	Stops open mode

- **Open mode** is automatically switched on when:
  - No disk space for a temporary copy of the edited file.
  - No / wrong terminal variable $TERM set.

vi - Modi



vi - Entry

Entry

`vi`	Starts the `vi` with an empty file
`vi` *`FILE`*	Starts the `vi` with an existing *`FILE`*

- During editing the file content is stored in a **buffer** in directory `/tmp` (or as a **hidden file** `.`*`FILE`*`.swp`).
- The **original file will be overwritten** not before exit of `vi` or explicit saving of the file.

vi - Exit

Exit

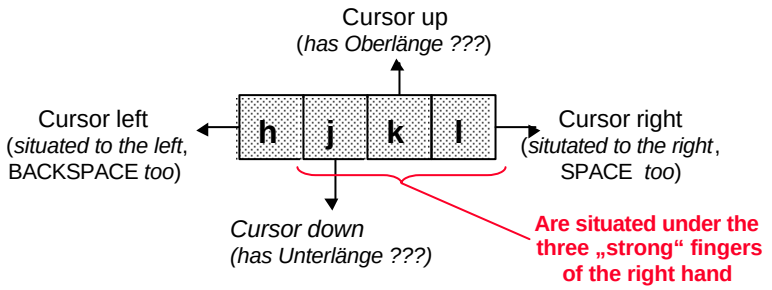
`:q`	Exits `vi`, if no changes have been done (**quit**).
`:q!`	As above, any changes are ignored (**quit**).
`:wq`	**Write** (changed) file and exit `vi` (**quit**).

Save file during edit ???

`:w`	**Write** changed text to file, but don‘t exit `vi`.

vi - Move

Standard Movements



vi - Move

Remark

- The cursor keys are often installed ??? with the corresponding `vi` commands and work even in edit mode.
- If the cursor keys are used in edit mode, **„invisible“ mode changes** permanently take place, that are not recognized.
- Therefore an urgent request:

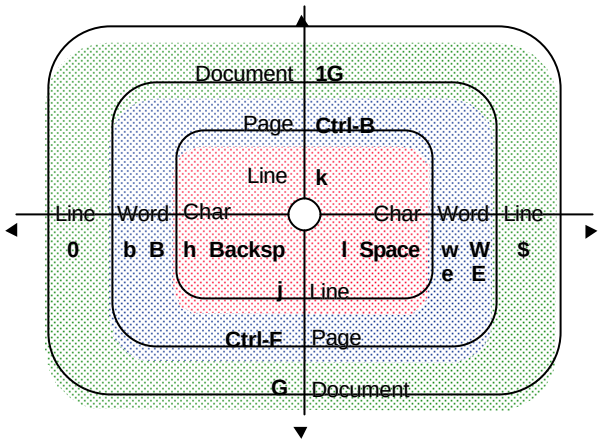
Please **cover** the cursor keys during the 1. training hour **with a piece of paper** and just use the `vi` commands to move the cursor!

vi - Move

Further cursor movements

`w W`	Wordwise forward (**word**)
`b B`	Wordwise back (**backword**)
`e E`	Wordwise to the end of word (**endword**)
`0 $`	To start / end of line (*zero!*)
`Ctrl-F`	One page down (**forward**)
`Ctrl-B`	One page up (**backward**)
`1G G` *`NN`*`G`	Skip to start of file / end of file / line number *NN* (**go**)

vi - Move



Exercise 20

- **Open** a copy of `poem` with the `vi`.
- Try out all **movement commands** (*please cover the cursor keys before!*).
- What is the **difference** between the commands
  - `w` and `W`?
  - `b` and `B`?
  - `e` und `E`?
- **Exit** the `vi` once without and once with saving the file.

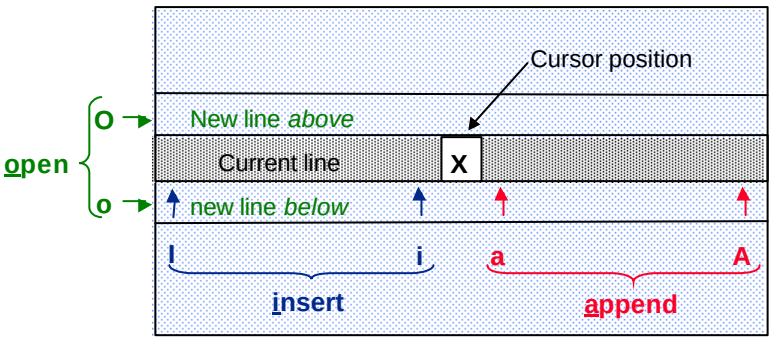
vi - Edit

Insert text

`a`	`A`	**Appends** text **after** the cursor position (`a`) resp. at the end of the current line (`A`).
`i`	`I`	**Inserts** text **before** the cursor position (`i`) resp. at the start of the current line (`I`).
`o`	`O`	**Opens** a **new (empty) line** below (`o`) resp. above (`O`) the current line, where text may inserted then.

vi - Edit

Insert Text



vi - Edit

- **Good working policy**
  - **To hit** `ESC` **before each command** works, but `ESC` is then often hit without any sense.
  - It is much better to close each **edit command** with `ESC` as soon as it is finished.
    - I.e. to do a sort of „**bracing**“ of the text input.
    - Please avoid the cursor keys during edit mode (*why?*).
  - The **current mode** may be seen in the **status line** (if the option `showmode` is active).
    - After some inital period you have the current mode **in your head** without thinking about that.

vi - Edit

- The commands to
  - **Delete** (`d`=**delete**)
  - **Change** (`c`=**change**)
  - **Save** (`y`=**yank**)

  are always **combined** with a movement command.
- I.e. a lot of possibilities for text editing are available without having **to mark the text before** (with cursor movements).

vi - Edit

Delete (and save) text (cut & paste)

`x`	Delete character under cursor (**crossout**).
`dw`	**Delete** from cursor to next start of **word**.
`d$`	**Delete** from cursor to end of line.
`dd`	**Delete** current **line**.
`dG  d1G`	**Delete** from current line to end / start of file (**go**).
`d`*`MOVE`*	**Delete** according to *`MOVE`* command.
`p  P`	**Insert** deleted Text (**put, paste**).

vi - Edit

Save text and insert it again (copy & paste)

`yb`	Save from cursor to previous word end (**yank backword**).
`y0`	Save from cursor to start of line (**yank**).
`3yy`	Save next 3 lines (**yank line**).
`y`*`MOVE`*	Save according to *`MOVE`* command (**yank**).
`p P`	**Insert** saved text **before** / **after** the current character or line (**put, paste**), depending on wether a ??? character or a line has been saved..

vi - Edit

Replace text

`R`	**Replace** from current character by keyed in text.
`r`	**Replace** current character by 1 char (*no* ESC*!*).
`cw`	**Change** current **word** from cursor to word end.
`c$`	**Change** from cursor to end of line.
`cc`	**Change** current **line**.
`cG c1G`	**Change** from current line to end / start of file (**go**).
`c`*`MOVE`*	**Change** according to *`MOVE`* command.

vi - Edit

Undo changes

`u`	**Undo last** change, no matter how big it was. May be done **just one time** in `vi`, `u` again **undos the former undo**!
`U`	**Undo all** changes in current line before it is left.

`vim` undoes an arbitrary number of changes!

`Ctrl-R`	**Undo last** undo operation in `vim` (**redo**, *has no function in* `vi`).

vi - Edit

Repetition of commands

- A command may be **repeated** by keying in a **repetition number** in front of it.

`8x`	(*delete 8 characters*)
`3dd`	(*delete 3 lines*)
`5cw` *TEXT...* `ESC`	(*change 5 words*)

- The **last edit command** may be **repeated** by a dot „`.`“.

`dd`	`. . .`	(*delete line*)
`P`	`. . .`	(*paste text*)
`cw` *TEXT...* `ESC`	`. . .`	(*change word*)

Exercise 21

- **Open** the file `poem` with the `vi`.
- **Change** the text by giving yourself **practical change exercises**.
  - Try out all insert, change, delete, save, copy and replace commands.
  - Undo the changes too.
- **Exit** the `vi` without saving the changes.
- Repeat the exercise and this time **save** the text file during editing and on exit.

vi - Search

Search

`/`*TEXT*	Search for string *TEXT* **forward** (to end of text).
`?`*TEXT*	Search for string *TEXT* **backward** (to start of text).
`n`	Search again in former direction (**next**).
`N`	Search again in oppsite direction (**next**).

- *TEXT* may be a so called **regular expression**.
- Search may even be used as movement command with the edit commands `c d y > < !`.

vi - Search & Replace

Search & replace

- Search whole file (`1,$`) for *TEXT* and replace it by string *SUBST* (**substitute**, **global**):
  `:1,$s/TEXT/SUBST/g`
- In front of `s` (**substitute**) a **line number** *NR* or a **line number range** *START,END* may be given to restrict the substitute operation to a line range, abbreviations for it are:
  `.` = **current line**
  `$` = **last line** of file
  `%` = **last line** of file (*corresponds to* `1,$`)
- Without a range just the **current line** will be treated ???.

vi - Search & Replace

Search & replace

- **Slashes** „`/`“ surround *TEXT* and *SUBST*.
- If a **slash** „`/`“ occurs in *TEXT* or *SUBST*,
  it has to be **protected** (quoted) by a „`\`“ in front of it.
  `:s/\/bin/\/usr\/bin/g`
- Alternatively a different separator may be used (e.g „@“):
  `:s@/bin@/usr/bin@g`
- *TEXT* may be a **regular expression**, in *SUBST* some parts of *TEXT* may be reused (*extremly powerful*).

vi - Search & Replace

Search & replace

- If `g` (**global**) is missing at the end, the substitution will take place just **1 time** per line (at first occurence of *TEXT*).
- If a number *n* (max. 512) at the end is given, just the ***n*th occurence** of *TEXT* will be substituted.

Confirm replacement

- To search the whole file (`1,$`) for *TEXT*, **mark** fitting texts (with `^^^` below) and replace them by *SUBST* if **confirmed** with `y` (**yes**) :

  `:1,$s/`*TEXT*`/`*SUBST*`/g`**`c`**

Exercise 22

- **Search** in `vi` through the file `poem` forward and backward for texts (e.g. `made`).
- **Replace** some of these texts with the `vi` commandos for changing, replacing and overwriting (*may be repeated by command „* `.` *“*).
- **Replace** some texts with the `ex` commands for search & replace (with and without confirmation,, e.g. `made` by `marmelade`).

vi - Options

- The `vi` configuration is done by **options**.
  - They have meaningful **English names** (`showmode`).
  - Many of them may be **shortened** (`showmode` be `sm`)
- To **set** an option:
  `:set` *`OPTION`*
- To **reset** an option:
  `:set` **`no`***`OPTION`*

vi - Options

- Besides on / off options (flags) there are **numerical** and **text** options, e.g.:
  `:set report=10`
  `:set dir=/tmp`
- The values of **changed options** are displayed by
  `:set`
- The values of **all options** are displayed by
  `:set all`

vi - Options

- The **most important options**:

`ignorecase`	Ignore UPPER / lower case during search
`list`	Show tabulators and newlines
`number`	Show lines with leading **line numbers**
`report=`*`NN`*	Show number of changed lines from *`NN`* lines on in status line
`showmode`	Show **vi mode** in status line
`wrapmargin=`*`NN`*	Insert line breaks automatically *`NN`* characters before end of line (`0`=*no break*)
`wrapscan`	Continue search around text start / end

vi - Options

- **Options** for **programmers**:

`autoindent`	Indent automatically (`Ctrl-D` / `T`)
`showmatch`	Show opening parentheses on input of closing one
`tabstop=`*`NN`*	Set tabulator width *`NN`*
`shiftwidth=`*`NN`*	Set tabulator width *`NN`* for later indenting with „`>>`“ and „`<<`“

- **Exception** (*to be used without* `set`):

`:syntax on`	Swítch **syntax coloring** on (`vim`!)

vi - Configuration File

- On each `vi` call the **configuration file** `~/.exrc` is read. It defines the **basic setup**.
  - For `vim` the file is named `~/.vimrc`.
- In that file **any** `ex` **command** may be given (*omit leading colon*):

```
set wrapscan
set report=1
set tabstop=4
set showmode
```


Exercise 23

- Change some **options directly** in the `vi`.
  - Try out the changed behaviour.
- Change some options in the **configuration file** `~/.exrc` (resp. `~/.vimrc`) to your needs (*with the* `vi`*!*).
  - Try out the changed behaviour.

UNIX-Editoren - vi

Table of Contents (2)

vi - Edit

Often used commands

`.`	Repeat last edit command (*very helpful!*).
`~`	Switch UPPER / lower case of current character.
`xp`	Exchange 2 characters (**crossout** + **put**).
`ddp`	Exchange 2 lines (**delete line** + **put**)
`dwwP`	Exchange 2 words (**delete word** + **word** + **Put**).
`deep`	Same (start at blank before first word)
`X`	Same as BACKSPACE (**crossout**)

vi - Edit

Special commands

`J`	**Join** two lines.
`%`	Jump to the corresponding parentheses (*for* `{[()]}`, *even if* ***nested***).
`>>  <<`	**Indent** current line in / out.
`>`*`MOVE`*	Indent text selected by *`MOVE`*.
`<`*`MOVE`*	Indent out text selected by *`MOVE`*.
`:!`*`CMD`*	Execute shell command (**shell escape**).
`:r!`*`CMD`*	Insert shell command result (**read**).
`:r `*`FILE`*	Insert file content (**read**).

vim - Visual Mode

Selecting text in visual mode (`vim` speciality)

`v`	Select text block **character by character**
`V`	Select text block **line by line**
`Ctrl-V`	Select text block **columnwise**

- The following commands may be applied to the selected text then

  `c d y > < ~ !`

vi - Move

Special Moves and Searches

`H M L`	Set cursor on start of the first (**home**) / **middle** / **last** line of the screen.
`tC TC`	Search character *C* forward / backward (set cursor **before** / **behind** the ch., **to**).
`fC FC`	Search character *C* forward / backward (set cursor **on** the found character, **find**).
NN`|`	Skip to column *NN* (*Pipe!*).
`^`	Set cursor on 1. „true“ character of a line.

Exercise 24

- **Try out** the special commands of `vi` you learnt in that chapter:
  - **Often used** commands
  - **Concatenate** lines
  - Jump between **corresponding parentheses**
  - **Indent** lines in and out
  - **Visual mode** (*only in* `vim`)
  - Specal **movements** and **searches**

vi - Marks

Set marks and skip to them

m*C*	Set **mark** *C* (26 different ones a-z possible) to current line / character.
'*C*	Skip to **start of line** with mark *C*.
`*C*	Skip to **character** with mark *C*.
''	Skip to **start of line** before last movement command (back and forth).
``	Skip to **character position** before last movement command (back and forth).

vi - Buffer

Save to / insert from named buffers

"*C*y*MOVE*	**Copy** selected text to buffer *C* (26 different ones a-z possible) (**yank**).
"*C*d*MOVE*	**Cut** selected text to buffer *C* (**delete**).
"*C*p	Insert text in buffer *C* **after** current line / cursor position **(put)**.
"*C*P	Insert text in buffer *C* **before** current line / cursor position **(Put)**.

vi - Macros

- The `vi` allows to put **selfdefined commands** on certain keys with the help of **macros**.
- Macros (**mapping**) are defined by
  `:map` *`KEY CMD`*
  and deleted by
  `:unmap` *`KEY`*
- E.g. normally the cursor keys are imposed ??? with the corresponding `vi` commands by macros.

vi - Abbreviations

- The `vi` knows **abbreviations**, which means that some words (**during text input**) are converted into another text (*after keying in the space*).
- **Abbreviations** are defined by
  `:ab` *`SHORTCUT LONGTEXT`*
  and ar deleted by
  `:unab` *`SHORTCUT`*
  Example:

```
:ab kr Kind Regards
```

Exercise 25

- Try out to set **marks** and jump to them.
- Try out **named buffers**.
- Try out **macros**.
- Try out **abbreviations**.