Outline

- Introduction - history
- Command line basics – getting help
- File system
- ➢Working with files and directories
- More file handling
- The shell revisited
- Monitoring resources

Working with Files and Directories

Naming

- **Do not use spaces.**
  You can use - or _ instead
- **Do not begin the name with - (dash).**
  Commands treat names starting with - as options.
- **Stick with letters, numbers, . (dot), - (dash) and _ (underscore).**
  Many other characters have special meanings on the command line.
- **Meaningful name.**
- **File extension or not.**
  Is just a convention, in Linux file extensions are not necessary. Files contain bytes. However, two-part names are used to help keep different kinds of files apart.

Creating files: text editing

Text editor

- Tool to create and edit files.
- There is no best text editor; it depends on personal taste.
- Text-only text editors
  - Simplicity first:
    - nano
  - With a steep learning curve: (needed for sysadmins and great for power users)
    - vi, vim
    - emacs
- Graphical text editors
  - Gedit: general purpose GUI based text editor

Primitive text editing

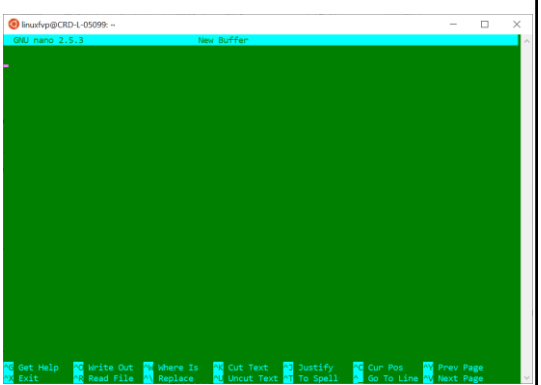
- Combine Redirection and Viewing
- use `cat` to direct stdin to a text file
  - `$ cat > my_text.txt`
  - Enter text.
  - To end the text input, press **Ctrl-D**.
- Check with `cat my_text.txt`
- Try adding another line of text to the existing file
  - `$ cat >> my_text.txt`

Primitive text editing

- Create an empty file
- `>filename`: Use redirection to create an empty file filename.
- `touch filename`: create an empty file, if the file does not exist yet.

nano

- Entering text: nano is a "modeless" editor. This means that all keystrokes, with the exception of Control and Meta sequences, enter text into the file being edited.
- Commands (lower part) are given by using the Control key (Ctrl, shown as ^) or the Meta key (Alt or Cmd, shown as M-).
  - A control-key sequence is entered by holding down the Ctrl key and pressing the desired key.
  - A meta-key sequence is entered by holding down the Meta key (normally the Alt key) and pressing the desired key.
- Manual: https://www.nano-editor.org/dist/v4/nano.pdf
- Cheat sheet: https://www.nano-editor.org/dist/latest/cheatsheet.html



vi

- Text-mode text editor available in all Linux systems.
- Created before computers with mice appeared.
- Very productive for power users.
- Check the web for tutorials:
  - https://upload.wikimedia.org/wikipedia/commons/d/d2/Learning_the_vi_Editor.pdf
  - ftp://ftp.vim.org/pub/vim/doc/book/vimbook-OPL.pdf

vi

- 2 basic modes of operation:
  - *command mode* and *editing mode*.
  - Within Command Mode, signals from the terminal are interpreted as editing commands.
  - Editing mode: letters typed at the keyboard are inserted into the editing buffer.
- Pressing **Esc** on the keyboard activates command mode.

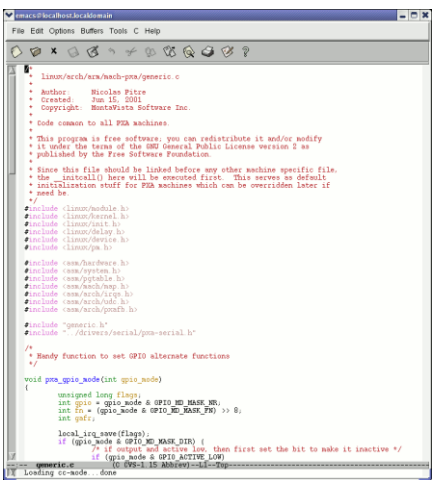
Key(s)	Function	Key(s)	Function
:w	Save	A	Append text after
:x	Save and exit	r	Replace text before cursor
:q	Quit	R	Replace text after cursor
I	Insert text after	i	Insert text before
P	Paste copied text	yy	Copy current line
a	Append text before	/[TEXT]	Search for the specified text

vi

- 2 modes
  - Input mode
    - ESC to back to cmd mode
  - Command mode
    - Cursor movement
      - h (left), j (down), k (up), l (right)
      - ^f (page down)
      - ^b (page up)
      - ^ (first char.)
      - $ (last char.)
      - G (bottom page)
      - :1 (goto first line)
    - Switch to input mode
      - a (append)
      - i (insert)
      - o (insert line after
      - O (insert line before)
- Delete
  - dd (delete a line)
  - d10d (delete 10 lines)
  - d$ (delete till end of line)
  - dG (delete till end of file)
  - x (current char.)
- Paste
  - p (paste after)
  - P (paste before)
- Undo
  - u
- Search
  - /
- Save/Quit
  - :w (write)
  - :q (quit)
  - :wq (write and quit)
  - :q! (give up changes)

Emacs

- Extremely powerful text editor features
- Great for power users
- Non standard shortcuts
- Much more than a text editor (games, e-mail, shell, browser).
- Some power commands have to be learnt.



Emacs

- $ emacs
- Cursor movement
  - ^f (forward one char.)
  - ^b (backward one char.)
  - ^a (begin of line)
  - ^e (end of line)
  - ^n (next line)
  - ^p (prev. line)
  - ^v (page up)
  - alt-v (page down)
- Deletion
  - ^d (delete one char)
  - alt-d (delete one word)
  - ^k (delete line)
- Paste
  - ^y (yank)
- Undo
  - ^/
- Load file
  - ^x^f
- Cancel
  - ^g
- Save/Quit
  - ^x^c (quit w/out saving)
  - ^x^s (save)
  - ^x^w (write to a new file)

Viewing files

Displaying file contents

Several ways of displaying the contents of files (without editing).

- `$ cat file1`
  displays the contents of the given file.
- `$ cat file1 file2 file3 ...` (concatenate)
  Concatenates and outputs the contents of the given files.
- `$ more file1`
  Display the output of a command or text file one page at a time.
  - Can also jump to the first occurrence of a keyword (`/` command).

Displaying file contents

- `$ less file1`
  - Does more than more (less is more ☺).
  - Doesn't read the whole file before starting.
  - Supports backward movement in the file (? command).
  - Search with `/`, next (`n` or `N`)
  - Press `q` to exit
- `$ display file1`
  Displays graphical file (simple image) (needs imagemagick)

The head and tail commands

- `$ head [-<n>] <file>`
  Displays the first <n> lines (or 10 by default) of the given file.
  Doesn't have to open the whole file to do this!
- `$ tail [-<n>] <file>`
  Displays the last <n> lines (or 10 by default) of the given file.
  No need to load the whole file in RAM! Very useful for huge files.
- `$ tail -f <file>` (follow)
  Displays the last 10 lines of the given file and continues to display new lines when they are appended to the file.
  Very useful to follow the changes in a log file, for example.

File manipulation

Move Copy Delete

File Manipulation

- For all file manipulation commands relative of absolute paths can be used (or just files in the current directory when no extra path specified)
- Most of the commands are intuitive – shortcuts of English names

Directories

- Create directories
  - The `mkdir` command is used to create directories
  - `$ mkdir dir1 dir2 dir3`
- Remove directories
  - The `rmdir` command removes directories
  - `$ rmdir dir1`
  - `rmdir` will only remove empty directories.
    To remove a non-empty directory, use
    `$ rm -r [DIRECTORY]` instead. (BE CAREFUL!)

Copy a file

- The `cp` command copies files and directories
- The default behavior will overwrite any existing file(s). The **-i** option overrides this behavior and prompts the user before overwriting the destination file.
- **syntax: `cp [OPTIONS] [SOURCE] [DESTINATION]`**
  - `$ cp <source_file> <target_file>`
    Copies the source file to the target.
  - `$ cp file1 file2 file3 ... dir`
    Copies the files to the target directory (last argument).
  - `$ cp -i` (interactive)
    Asks for user confirmation if the target file already exists
  - `$ cp -r <source_dir> <target_dir>` (recursive)
    Copies the whole directory.
  - `$ cp -v` (verbose)
    Displays what has been copied

Move or rename files

- Move or rename files and directories: `mv`
- The default behavior will overwrite any existing file(s).
- **syntax: `mv [OPTIONS] [SOURCE] [DESTINATION]`**
  - `$ mv old_name new_name`
    Renames the given file or directory.
  - `$ mv -i` (interactive)
    If the new file already exits, asks for user confirm
- The `mv` command can also be used to move or rename directories
  - `$ mv NewFiles/ OldFiles/`
  - -r option is not necessary

Remove files

- The rm command removes files.
- `$ rm file1 file2 file3 ...`
  Removes the given files.
- `$ rm -i` (interactive)
  Always ask for user confirmation.
- `$ rm -r dir1 dir2 dir3` (recursive)
  Removes the given directories with all their contents.