#+Title Emacs config with explanations

Emacs config with explanations

General UI config

- Sets the default font size in a variable called `emacs/efault-font-size`
- Inhibits the starting message (you can tell it's enabled due to the tag **t**).
- Enables scroll bar and menu bar (which should be enabled by default) with the tag **1**.
- Loads the theme **modus-operandi-tritanopia**, the `'` indicates that a list is to follow.
  - To preview themes, use **M-x load-theme** and then just scroll using up and down arrow keys to select a theme to preview.
- Sets a global key **option - 3** (on macOS) - to bypass the meta key conflict. Disable if on any other system.

Code

```
(defvar emacs/default-font-size 150)

(setq inhibit-starting-message t)

(scroll-bar-mode 1) ;; enable scroll bar
(menu-bar-mode 1) ;; enable menu bar

(load-theme 'modus-operandi-tritanopia t)

(global-set-key (kbd "M-3") (lambda () (interactive) (insert "#")))
```

Font Configuration

- Sets the default font and calls the height (*size*) from the already defined variable `emacs/default-font-size`.
- Sets a fixed pitch face, for org-mode that is defined with height `150`.
- Sets a variable pitch face, for org-mode that is defined with height `200` and weight `regular` (which means not bold).

BEAR IN MIND:

- The font I am using is *Fira code retina* which may not be installed on your computer. Please follow this link to install it if you wish: Fira Code Retina Github.
  - For installation instructions for macOS:
    - Download the zip file from their github.
    - Extract the file.
    - Open the TTF folder.
    - Select all font files.
    - Right click and select Open.

    - Select "Install Font".

- The second font I am using for the variable pitch face is *Cantarell* - a Google font which may also not be installed on your computer. Please follow this link if you wish to install it: Cantarell Google Fonts

Code

```
;;  (set-face-attribute 'default nil :font "Fira Code Retina" :height
emacs/default-font-size)

(set-face-attribute 'default nil :height emacs/default-font-size)

  ;; Set the fixed pitch face
;;  (set-face-attribute 'fixed-pitch nil :font "Fira Code Retina" :height
150)

  ;; Set the variable pitch face
;;  (set-face-attribute 'variable-pitch nil :font "Cantarell" :height 200
:weight 'regular)
```

Initialise Package Sources

- Require package.
- The locations of archives that Emacs will look into for any packages that you wish to install.
    - Melpa
    - Org
    - Elpa
- Initialise packages, and then unless the package is in the archive, refresh the contents (essentially, refresh your *installed* packages).
- Enable the package `use-package` and enable it all the time by using the tag **t** .
- Initialise the **which-key** package, which is a useful package that shows the possible commands when you press any hotkeys, such as **C-x** / **M-x** or anything else. The `which-key-idle-delay` refers to how many *seconds* to wait before displaying the possibilities menu, in this case set to **1 second**.
- Initialise the use of the **counsel** package which enables easy to use shortcuts, such as finding your file `C-x C-f` and other useful shortcuts.
- Initialise the use of the **ivy** package which has one of the best functions **C-s** which corresponds to *swiper* a useful programme that can run to identify any text in your file and further shortcuts. To enable `ivy-mode`, a tag of **1** is set.
- Another function is seen here, to define `efs/org-mode-setup` which we will see soon enough in our org-mode configuration. It just indents our org files, sets the variable font and wraps text.

Code

```
(require 'package)

(setq package-archives '(("melpa" . "https://melpa.org/packages/")
                         ("org" . "https://orgmode.org/elpa/")
                         ("elpa" . "https://elpa.gnu.org/packages/")))

(package-initialize)
(unless package-archive-contents
 (package-refresh-contents))

;; Initialize use-package on non-Linux platforms
(unless (package-installed-p 'use-package)
   (package-install 'use-package))

(require 'use-package)
(setq use-package-always-ensure t)

(use-package which-key
  :init (which-key-mode)
  :diminish which-key-mode
  :config
  (setq which-key-idle-delay 1))

(use-package counsel
  :bind (("M-x" . counsel-M-x)
         ("C-x b" . counsel-ibuffer)
         ("C-x C-f" . counsel-find-file)
         :map minibuffer-local-map
         ("C-r" . 'counsel-minibuffer-history))
  :config
  (setq ivy-initial-inputs-alist nil)) ;; Don't start searches with ^

(use-package ivy
  :diminish
  :bind (("C-s" . swiper)
         :map ivy-minibuffer-map
         ("TAB" . ivy-alt-done)
         ("C-l" . ivy-alt-done)
         ("C-j" . ivy-next-line)
         ("C-k" . ivy-previous-line)
         :map ivy-switch-buffer-map
         ("C-k" . ivy-previous-line)
         ("C-l" . ivy-done)
         ("C-d" . ivy-switch-buffer-kill)
         :map ivy-reverse-i-search-map
         ("C-k" . ivy-previous-line)
         ("C-d" . ivy-reverse-i-search-kill))
  :config
  (ivy-mode 1))

(defun efs/org-mode-setup ()
  (org-indent-mode)
  (variable-pitch-mode 1)
  (visual-line-mode 1))
```

Make E-macs smoother and faster

- You may have noticed e-macs running a bit slower, these following changes should help it back to its original snappy pace. I also commented out some lines that I feel were not necessary and were worth taking it out just for that bit more of performance.

```
(setq fast-but-imprecise-scrolling t)
(setq redisplay-skip-fontification-on-input t)
(setq inhibit-compacting-font-caches t)

(setq gc-cons-threshold 100000000) ;; 100MB
(setq read-process-output-max (* 1024 1024)) ;; 1MB (helps LSP too)

(setq org-fontify-whole-heading-line nil) ;; only color the text, not
the full line
(setq org-fontify-quote-and-verse-blocks nil) ;; skip extra styling on
quotes
```

Org-mode configuration

- At last, Org-mode, one of the best modes in Emacs.
- Define the function `efs/org-font-setup`:
  - to replace the listed hyphen with the a dot.
  - Then for each heading, subheading, etc, set levels to make it easier to assign fonts.
- Initialise the **org** package (enabled by default, but doesn't hurt to have it in).
  - set a *hook* (which is something that runs anytime you open an certain file, in this case an org file).
    - this hook just executes the function mentioned, `efs/org-mode-setup`. and then replaces the elipsis with a **much** nicer looking down-arrow.

Org-agenda

- Org-agenda is amazing feature of org-mode allowing you to track all your tasks and more.
  - Here, we add all the agenda files that we feel we need.

Org-habit

- If you have tasks that you do every day, setting the task to done would then mark it as completed and over time, a graph would form that you can view in `org-agenda`.

Org-todo

- You can set tasks, with the tags `TODO`, `DONE`, but I felt that having the tag `NEXT` would help and so that was added, you can add however much you need.

Org-refile

- To make things even more amazing, `org-refile` enables you to note down a task in any old org file and then refile into another, more structured org file that is added to the `org-refile-targets` list.
  - The `:maxlevel . 2` indicates that you can refile even to the subheadings of that document, whereas `:maxlevle . 1` indicates that you can only refile to the headings.
- And to not forget, an automated command that saves all org-buffers, after refiling.

Org-tags

- A bunch of tags that you can assign to your tasks to make your life easier and more

organised.

Org-capture

- `org-capture-templates` can be accessed through any old buffer and are there in the case of any idea popping up in your head. All you have to do is `M-x org-capture` and a little menu should pop up with prompts.

Org-bullets

- Sets up **much** nicer looking bullets.

Code

```
  (defun efs/org-font-setup ()
    ;; Replace list hyphen with dot
    (font-lock-add-keywords 'org-mode
                            '(("^ *\\([-]\\) "
                               (0 (prog1 () (compose-region (match-
beginning 1) (match-end 1) "•"))))))

    ;; Set faces for heading levels
;;    (dolist (face '((org-level-1 . 1.2)
;;                    (org-level-2 . 1.1)
;;                    (org-level-3 . 1.05)
;;                    (org-level-4 . 1.0)
;;                    (org-level-5 . 1.1)
;;                    (org-level-6 . 1.1)
;;                    (org-level-7 . 1.1)
;;                    (org-level-8 . 1.1)))
;;      (set-face-attribute (car face) nil :font "Cantarell" :weight
'regular :height (cdr face)))

     ;; Ensure that anything that should be fixed-pitch in Org files
appears that way
    (set-face-attribute 'org-block nil :foreground nil :inherit 'fixed-
pitch)
    (set-face-attribute 'org-code nil   :inherit '(shadow fixed-pitch))
    (set-face-attribute 'org-table nil   :inherit '(shadow fixed-pitch))
    (set-face-attribute 'org-verbatim nil :inherit '(shadow fixed-pitch))
    (set-face-attribute 'org-special-keyword nil :inherit '(font-lock-
comment-face fixed-pitch))
    (set-face-attribute 'org-meta-line nil :inherit '(font-lock-comment-
face fixed-pitch))
    (set-face-attribute 'org-checkbox nil :inherit 'fixed-pitch))


  (use-package org
    :hook (org-mode . efs/org-mode-setup)
    :config
    (setq org-ellipsis " ▾")

  (setq org-agenda-start-with-log-mode t)
  (setq org-log-done 'time)
  (setq org-log-into-drawer t)

  (setq org-agenda-files
        '("~/Orgfiles/Tasks/Tasks.org"
          "~/Orgfiles/Tasks/Home.org"
          "~/Orgfiles/Tasks/Uni.org"
          "~/Orgfiles/Journal/Journal.org"))

   (require 'org-habit)
    (add-to-list 'org-modules 'org-habit)
    (setq org-habit-graph-column 60)

    (setq org-todo-keywords
        '((sequence "TODO(t)" "NEXT(n)" "|" "DONE(d!)")))

    (setq org-refile-targets
```

```
    (setq org-refile-targets
      '(("~/Tasks/Tasks.org" :maxlevel . 2)
        ("~/Tasks/Home.org" :maxlevel . 2)
        ("~/Tasks/Uni.org" :maxlevel . 2)
        ("~/Tasks/Journal.org" :maxlevel . 2)))

    ;; Save Org buffers after refiling!
    (advice-add 'org-refile :after 'org-save-all-org-buffers)

  (setq org-tag-alist
      '((:startgroup)
         ; Put mutually exclusive tags here
         (:endgroup)
         ("assignment" . ?a)
         ("research" . ?r)
         ("assessment" . ?A)
         ("quiz" . ?q)
         ("lab" . ?l)
         ("habit" . ?h)))

  (setq org-capture-templates
      `(("t" "Tasks / Projects")
        ("tt" "Task" entry (file+olp "~/Orgfiles/Tasks/Tasks.org" "Quick-
capture")
             "* TODO %?\n  %U\n  %a\n  %i" :empty-lines 1)

        ("j" "Journal Entries")
        ("jj" "Journal" entry
             (file+olp+datetree "~/Orgfiles/Journal/Journal.org")
             "\n* %<%I:%M %p> - Journal :journal:\n\n%?\n\n"
             ;; ,(dw/read-file-as-string "~/Orgfiles/Journal/Journal.org")
             :clock-in :clock-resume
             :empty-lines 1)

        ("m" "Metrics Capture")
        ("mh" "Health" table-line (file+headline
"~/Orgfiles/Health/Health.org" "Health")
         "| %U | %^{Type}  | %^{Food}  | %^{Calories} | %^{Notes}  |"
:kill-buffer t)))

  (efs/org-font-setup))


  (use-package org-bullets
    :after org
    :hook (org-mode . org-bullets-mode)
    :custom
    (org-bullets-bullet-list '("◉" "○" "●" "○" "●" "○" "●")))
```

Key-bindings and evil-mode

Icons

- To ensure our modeline looks good, the following will enable a doom-like modeline.
    - **BEAR IN MIND**: The first time running this you'll need to run the following command: `M-x all-the-icons-install-fonts`

1. Code:

```
   ;;  (use-package all-the-icons)

   ;;  (use-package doom-modeline
   ;;  :init (doom-modeline-mode 1)
   ;;  :custom ((doom-modeline-height 15)))
```

Code

```
(use-package evil
    :init
    (setq evil-want-integration t)
    (setq evil-want-keybinding nil)
    (setq evil-want-C-u-scroll t)
    (setq evil-want-C-i-jump nil)
    :config
    (evil-mode 1)
    (define-key evil-insert-state-map (kbd "C-g") 'evil-normal-state)
    (define-key evil-insert-state-map (kbd "C-h") 'evil-delete-backward-
char-and-join)

    (evil-set-initial-state 'messages-buffer-mode 'normal)
    (evil-set-initial-state 'dashboard-mode 'normal))

  (use-package evil-collection
    :after evil
    :config
    (evil-collection-init))
```

IDE-like experience

- Any code in the source blocks acts just like it would in a normal c file open in emacs.
- Initialise the package `Lsp-ui` and appear the menu at the bottom and runs a hook that executs `lsp-ui-mode` anytime an `lsp-mode` file is opened.
- Initialise the package, `cc-mode` which is used to compile c files.
- Initialise the package `company` which is very useful for auto completions.
    - To make it look even nicer `company-box` is used which adds a nice UI for the autocompletion menu in the `company` package.
- Initialise the `smartparens` which automatically closes your parenthesis like all types of brackets.
- Initialise package `yasnippet` and runs a hook that executes `yas-minor-mode` any time a c file is opened in Emacs.

  BEAR IN MIND: When using `yasnippet` for the first time, ensure that you instal all of the snippets available. This is done through running the following.

    - `M-x package-install RET yasnippet-snippets RET`

Code

```
(setq org-src-tab-acts-natively t) ;; (Makes the src code blocks act like
a normal c file)

(use-package lsp-ui
  :hook (lsp-mode . lsp-ui-mode)
  :custom
  (lsp-ui-doc-position 'bottom))

(use-package cc-mode
  :ensure nil
  :mode ("\\.c\\'" . c-mode)
  :hook (c-mode . lsp-deferred)
  :config
  ;; optional: 2-space indentation
  (setq c-basic-offset 2))

(use-package company
  :after lsp-mode
  :hook ((lsp-mode . company-mode)
         (c-mode . company-mode)) ;; add company-mode to c-mode explicitly
  :bind (:map company-active-map
              ("<tab>" . company-complete-selection)
         :map lsp-mode-map
              ("<tab>" . company-indent-or-complete-common))
  :custom
  (company-minimum-prefix-length 1)
  (company-idle-delay 0.0))

(use-package company-box
  :hook (company-mode . company-box-mode))

(use-package smartparens
  :hook ((prog-mode . smartparens-mode))
  :config
  (require 'smartparens-config))

(use-package yasnippet
  :hook ((c-mode . yas-minor-mode)))
```

Org-babel

- Loads the `ob-C.el.gz` file just to ensure everything works fine.
- Loads the languages that you need, in my case **C**.
    - **BEAR IN MIND**: this is case-sensitive, I spent the better part of an hour trying to get this to work only to realise that *c* is not registered but **C** was.
- `org-structure-template-alist` will save your hand in Emacs, especially if you add alot of source code blocks in your files. Writing down `<el + TAB` will just automatically add a source code block in emacs-lisp. *highly recommended that you add any of your most used languages here*. **BEAR IN MIND**: The shortcut that you use may be in conflict with other shortcuts, e.g. `<c` conflicted with centering text / commenting text.
- `org-tangle` - to automatically tangle emacs config file when saved, and instead of always asking to save the file, add the tag **nil** . Then add a hook, that executes that function when saving this emacs config.

Code

```
(require 'ob-C)  ;; loads ob-C.el.gz transparently

(org-babel-do-load-languages
 'org-babel-load-languages
 '((C . t)))

(require 'org-tempo)

(add-to-list 'org-structure-template-alist '("code" . "src C"))
(add-to-list 'org-structure-template-alist '("el" . "src emacs-lisp"))
(add-to-list `org-structure-template-alist `("sh" . "src shell"))

;; Automatically tangle our Emacs.org config file when we save it
    (defun efs/org-babel-tangle-config ()
      (when (string-equal (buffer-file-name)
                          (expand-file-name "~/.emacs.d/Emacs.org"))

        ;; Dynamic scoping to the rescue
        (let ((org-confirm-babel-evaluate nil))
          (org-babel-tangle))))

    (add-hook 'org-mode-hook (lambda () (add-hook 'after-save-hook
#'efs/org-babel-tangle-config)))
```

Compiling C-files

- Define a function that compiles the file using `gcc -Wall`. Then add a hook that executes the function whenever the `C-c C-c` shortcut is pressed in a `c` file, to compile it.

Code

```
(defun my-c-compile ()
  "Compile current C file."
  (interactive)
  (let ((file (buffer-file-name)))
    (compile (format "gcc -Wall -O2 -o %s %s"
                     (file-name-sans-extension file)
                     file))))

(add-hook 'c-mode-hook
          (lambda ()
            (local-set-key (kbd "C-c C-c") 'my-c-compile)))
```

Term-mode

- Code

```
   (use-package eterm-256color
     :hook (term-mode . eterm-256color-mode))
```

V-term

1. Code

```
(use-package vterm
  :commands vterm
  :config
  (setq term-prompt-regexp "^[^#$%>\n]*[#$%>] *")  ;; Set this to
match your custom shell prompt
  ;;(setq vterm-shell "zsh")                       ;; Set this to
customize the shell to launch
  (setq vterm-max-scrollback 5000))
```

Effortless file management

Keyboard Shortcuts

1. Emacs/ Evil
   - `n / j` - next line
   - `p / k` - previous line
   - `j / J` - jump to file in buffer
   - `RET` - select file or directory
   - `^` - go to parent directory
   - `S-RET / g O` - Open file in “other” window
   - `M-RET` - Show file in other window without focusing (previewing files)
   - `g o (dired-view-file)` - Open file but in a “preview” mode, close with q
   - `g / g r` Refresh the buffer with revert-buffer after changing configuration
2. Marking a file
   - `m` - Marks a file
   - `u` - Unmarks a file
   - `U` - Unmarks all files in buffer
   - `*t / t` - Inverts marked files in buffer
   - `% m` - Mark files in buffer using regular expression
   - `*-` Lots of other auto-marking functions
   - `k / K` - “Kill” marked items (refresh buffer with g / g r to get them back)
3. Copying and renaming files
   - `C` - Copy marked files (or if no files are marked, the current file)
   - Copying single and multiple files
   - `U` - Unmark all files in buffer
   - `R` - Rename marked files, renaming multiple is a move!
   - `% R` - Rename based on regular expression: ^test , old-\&
4. Deleting files
   - `D` - Delete marked file
   - `d` - Mark file for deletion

- `x` - Execute deletion for marks
- `delete-by-moving-to-trash` - Move to trash instead of deleting permanently

5. Creating and extracting archives
   - `Z` - Compress or uncompress a file or folder to (.tar.gz)
   - `c` - Compress selection to a specific file
   - `dired-compress-files-alist` - Bind compression commands to file extension
6. Other common operations
   - `T` - Touch (change timestamp)
   - `M` - Change file mode
   - `O`- Change file owner
   - `G` - Change file group
   - `S` - Create a symbolic link to this file
   - `L` - Load an Emacs Lisp file into Emacs

Keeping folders clean

- Using package `no-littering`

```
;; NOTE: If you want to move everything out of the ~/.emacs.d folder
;; reliably, set `user-emacs-directory` before loading no-littering!
;(setq user-emacs-directory "~/.cache/emacs")

(use-package no-littering)

;; no-littering doesn't set this by default so we must place
;; auto save files in the same path as it uses for sessions
(setq auto-save-file-name-transforms
      `((".*" ,(no-littering-expand-var-file-name "auto-save/") t)))
```

Projectile and magit

To make editing on github much easier

Projectile

- Use the package `projectile`, and then bind the `C-c p` shortcut as a prefix key.
  - Essentially, `C-c p` opens the map, where you can access the projectile keyboard shortcuts.

```
(use-package projectile
:diminish projectile-mode
:config (projectile-mode)
:custom ((projectile-completion-system 'ivy))
:bind-keymap
("C-c p" . projectile-command-map)
:init
;; NOTE: Set this to the folder where you keep your Git repos!
(when (file-directory-p "~/Projects/Emacs")
  (setq projectile-project-search-path '("~/Projects/Emacs")))
(setq projectile-switch-project-action #'projectile-dired))

(use-package counsel-projectile
:config (counsel-projectile-mode))
```

Magit

- To enable and improve git intergation into emacs, `magit` is required.

```
  (use-package magit
    :ensure t
    :bind (("C-x g" . magit-status)))

  (use-package forge
:after magit)
```

Export options

PDFs and latextopdf

1. 1. On macOS
    - Install a LaTeX distribution:
        - macOS doesn't come with LaTeX by default. You have two main options:
            - MacTeX (recommended full distribution): Download: https://tug.org/mactex/ *(This includes pdflatex, xelatex, and all necessary packages)*. After installation, make sure /usr/texbin or /Library/TeX/texbin is in your PATH.
            - BasicTeX (lightweight version): Smaller download, but you may need to install extra packages as you export.
                - Install via Homebrew:

```
brew install --cask basictex
sudo tlmgr update --self
sudo tlmgr install collection-latexextra
```

2. Configure E-macs orgmode for PDF support

```
;; Enable LaTeX PDF export in Org
(require 'ox-latex)

;; Use xelatex for better font support (optional)
(setq org-latex-pdf-process
      '("xelatex -interaction=nonstopmode -output-directory=%o %f"
        "xelatex -interaction=nonstopmode -output-directory=%o %f"))
```