Example Post

Contents

Contents

1 The Problem with emacs on Windows

TECMACS – the software emacs originates from – was written in https://www.jwz.org/doc/emacs-timeline.html1976. Given that it has been developed ever since, it is no wonder it is available for all major operating systems. Nonetheless, some important functions depend on packages which are not natively available on Windows, so by default, an emacs installation on Windows falls short of these.

This should be one more reason [Link] for you to consider changing to an open-source operating system such as Linux. Please do not just gloss over this point. Next to security reasons, user-configurability and extendability, there are compelling arguments – both political and ethical – why you should not support big-tech companies. Plus there is https://en.wikipedia.org/wiki/Tux_(mascot)Tux, the mascot of Linux. And who does not want a penguin as their mascot?

That being said, some people depend on Windows if they like it or not (although many people only think this is the case, especially in academia [Link]), and if they want to use an open-source text editor instead of Word, this is a big plus already. So, let's start building emacs from source!

2 The Solution to the Problem

Since Windows is not capable of providing a satisfactory emacs experience but Linux is, we simply import all the necessary linux libraries to Windows. There are many ways to achieve this, but the one I prefer is to use `msys2`. `msys2` is an open-source software based on `cygwin` which does exactly what we need: Provide the necessary tools to build Linux software natively in Windows. So let's install it!

2.1 Installing `msys2`

Visit the https://www.msys2.org/website of `msys2` and download the latest Windows installer. Double-click it and follow the steps of the installation wizard. I installed it to `C:\msys64`.

Warning: Do *not* open any other of the `.exe` files. Following the below steps with the wrong executable opened *will* cause problems.

2.2 Configuring `msys2`

Go to `C:\msys64\home\<user>` and open `.bash.rc` with `notepad` by double-clicking on the file and, if necessary, selecting the notepad among the list of programs. Then, add the following code to the end of the file: language=bash,label= ,caption= ,captionpos=b,numbers=none if [(uname -o) == 'Msys']; then export PATH='echo PATH | awk -v RS=: -v ORS=: '/c/ next print' | sed 's/:*//'' fi This code snippet makes sure that issuing a command in `msys2` which happens to have the same name as a Windows command won't cause problems. Lastly, we want to make sure that `msys2` is recognized by Windows. We do that by setting a so-called `PATH` variable. With that, we just tell Windows to look for executables in that path. This way, we make sure that Windows finds `msys2`.

To do that, open the application launcher by pressing the Windows key, type in "environment variables" and click on the first result. If you have administrator rights, you can now click on "New" and add both the path to `msys2` and its subfolder `/bin`. For me, this was `C:\msys64\mingw64` and `C:\msys64\mingw64\bin`. If you do not have admin rights, you can just add the paths to your current path(s) by separating them by semicolons as in `C:\msys64\mingw64\bin;C:\msys64\mingw64\bin`.

And with that, `msys2` is configured! You can now back to `C:\msys64` and open `mingw64.exe`. Press on “environment variables” again, select “path” and click on “edit”.

3 Preparing to Install Emacs

Now we have the *means* to get all the libraries we need to build emacs, but we do not have a single one yet. Thus, the next step is to get them. Conveniently, `msys2` also uses `pacman`, the package manager which by default ships with Arch Linux, so it might even be familiar to you.

First of all, we want to make sure that our package manage is up-to-date: language=bash,label= ,caption= ,captionpos=b,numbers=none pacman -Syu The flags `S`, `y` and `u` stand for “Sync”, “refresh” and “sysupdate”, respectively. That being done, we install all tools we need to build emacs from source, starting with the most important ones: language=bash,label= ,caption= ,captionpos=b,numbers=none pacman -Sy –needed filesystem msys2-runtime bash libreadline libiconv libarchive libgpgme libcurl pacman ncurses libintl Now we’ll need to update `pacman` again: language=bash,label= ,caption= ,captionpos=b,numbers=none pacman -Su Finally, we will download and install the necessary libraries. Depending on your download speed and hardware, this will take quite a while. Perfect time to rethink whether you really need Windows. Remember you can also have two systems installed on your machine [Link] to try out Linux! language=bash,label= ,caption= ,captionpos=b,numbers=none pacman -Su autoconf autogen automake automake-wrapper diffutils git guile libgc libguile libidn-devel libltdl libnettle-devel libopenssl libp11-kit-devel libtasn1-devel libunistring make mingw-w64-x86$_6$4 − binutils mingw − w64 − x86$_6$4 − bzip2 mingw − w64 − x86$_6$4 − cairo mingw − w64 − x86$_6$4 − crt − git mingw − w64 − x86$_6$4 − dbus mingw − w64 − x86$_6$4 − expat mingw − w64 − x86$_6$4 − fontconfig mingw − w64 − x86$_6$4 − freetype mingw − w64 − x86$_6$4 − gcc mingw − w64 − x86$_6$4 − gcc − libs mingw − w64 − x86$_6$4 − gdk − pixbuf2 mingw − w64 − x86$_6$4 − gettext mingw − w64 − x86$_6$4 − giflib mingw − w64 − x86$_6$4 − glib2 mingw − w64 − x86$_6$4 − gmp mingw − w64 − x86$_6$4 − gnutls mingw − w64 − x86$_6$4 − harfbuzz mingw − w64 − x86$_6$4 − headers − git mingw − w64 − x86$_6$4 − imagemagick mingw − w64 − x86$_6$4 − isl mingw − w64 − x86$_6$4 − jansson mingw − w64 − x86$_6$4 − libffi mingw − w64 − x86$_6$4 − libgccjit mingw − w64 − x86$_6$4 − libiconv mingw − w64 − x86$_6$4 − libidn2 mingw − w64 − x86$_6$4 − libjpeg − turbo mingw − w64 − x86$_6$4 − libpng mingw − w64 − x86$_6$4 − librsvg mingw − w64 − x86$_6$4 − libsystre mingw − w64 − x86$_6$4 − libtasn1 mingw − w64 − x86$_6$4 − libtiff mingw − w64 − x86$_6$4 − libunistring mingw − w64 − x86$_6$4 − libwinpthread − git mingw − w64 − x86$_6$4 − libxml2 mingw − w64 − x86$_6$4 − mpc mingw − w64 − x86$_6$4 − mpfr mingw − w64 − x86$_6$4 − nettle mingw − w64 − x86$_6$4 − p11 − kit mingw − w64 − x86$_6$4 − pango mingw − w64 − x86$_6$4 − pixman mingw − w64 −

x86$_6$4 − poppler mingw − w64 − x86$_6$4 − winpthreads mingw − w64 − x86$_6$4 − xpm − nox mingw − w64 − x86$_6$4 − xz mingw − w64 − x86$_6$4 − zlib mingw − w64 − x86$_6$4 − jbigkit nano openssl pkgconf tar texinfo wget Having done this, we could in principle compile emacs from source. What we should not forget, though, is that emacs lives from packages – and they also have dependencies which we need to install. Of course, the code below might not suffice to cover all dependencies your packages need, but it takes care of the most important ones: `hunspell` to make spell-checking work and `poppler` to allow `pdf-tools` to open `.pdf` files. `sqlite` already is installed with the binaries above, so `org-roam` (and also `org-roam-ui`) will work. language=bash,label= ,caption= ,captionpos=b,numbers=none pacman -S mingw-w64-x86$_6$4 − hunspell mingw − w64 − x86$_6$4 − poppler mingw − w64 − x86$_6$4 − poppler − data Note that whilst only an english dictionary is available via `msys2`, you can add dictionaries of any language you like by putting them in `C:\msys64\mingw64\share\hunspell`. For a list of dictionaries to download, visit https://github.com/elastic/hunspell/tree/master/dictsthis page. Of course, you will need to adjust the emacs code, too, then. But this is a topic for [Link] another post.

For those who want to use emacs as a `LaTeX` editor or like having their formulas overlayed with `org-preview`, you should also install the `texlive` binaries. This installation should happen without the help of msys2; if you already have a LaTeX distribution installed, emacs will recognize it once it is configured. If not, you will need to install it. The easiest way I have found is to use an `.iso` https://www.tug.org/texlive/acquire-iso.htmlimage, but there are other possibilities as well.

If you think of using `mu4e` as your emacs client on Windows, this might be a problem because there is no `msys2` package for `mu`. You might try your luck with an https://github.com/msys2-unofficial/MSYS2-packages/tree/master/muunofficial package, though.

4 Building Emacs

Now we have done everything we need to start with the actual building process, so we are going to download the emacs source code from an official github repository: language=bash,label= ,caption= ,captionpos=b,numbers=none git clone http://git.savannah.gnu.org/r/emac

4.1 Configuring the Installation

Next up, we need to tell emacs what (not) to build. I have chosen to go with every sensible dependency to cover all needs: If you get an error message, you are likely in the wrong folder. Make sure the folder contains `autogen.sh`. You can do that by running `ls`, which lists the contents of the folder the console currently operates in.

4.2 Building Emacs

Now, we're finally ready to do the last step: Actually installing emacs! This will take at least 10 minutes, possibly up to 30. Just run the following code: language=bash,label= ,caption= ,captionpos=b,numbers=none make make install

4.3 Setting the Paths

With this, you have built your own emacs from source – Congratulations! To run emacs, we still need to put two files in the binary folder: `libdbus-1-3.dll` and `libgmp-10.dll`. You can download them https://www.exefiles.com/de/dll/libdbus-1-3-dll/here and https://www.dll-files.com/libgmp-10.dll.htmlhere, respectively. Just open the Windows explorer, type `%APPDATA%` in the folder bar and press `Enter`. You will now be in `C:\Users\<user>\AppData\Roaming`. Open the folder called `bin` and put the `.dll` file in there. If you now double-click on `runemacs.exe`, emacs should open.

5 Setting up an Emacs Daemon

Since emacs is not a Windows program, its performance on Windows is, to put it mildly, not out of this world. Especially if you have a large configuration file, it might take half a minute or even longer to get it started. This, of course, completely interferes with your workflow, so we'll need to find a way to deal with that. And the way to go is as simple as silently starting emacs at startup and letting it pop up whenever we want to edit a file in it. This running-in-the-background service is called a *server* or a *daemon*.

On Linux, setting up a daemon is as simple as typing in a single command. On Windows, on the other hand, things aren't as easy (as is often the case). To alleviate at least some of the despair, I have written some code which takes care of it and converted it to `.exe` files. Just download all the `.exe` files and put them in the folder in which the other emacs executables are. For me, it was `C:\Users\Vitus\AppData\Roaming\bin`. Since this folder by default is in the `Path` variable, we can skip the first step of the instructions on the github page and only need to follow the other 8 steps described on github; this will approximately take 5 minutes. And that's it. Now you have successfully installed the emacs daemon – Congratulations!

6 The Next Part of your Journey

If you have followed this guide up until now, your emacs is ready to use, but it will look unfamiliar and the keybindings will be very different from Windows, so you are likely to be completely overwhelmed. This is normal, and I was, too. For this, reason, as a next step, you will need to set up a configuration file which will make your life easier. If you now feel even more overwhelmed, no worries, this [Link] post is for you. Should you already know how setting up such a file works and just want the familiar keybindings, you might want to check out [Link] this post.

For those die-hards who already know all this, just put your configuration file in the folder which in turn contains `bin`. For me, that was `C:\Users\Vitus\AppData\Roaming`. Note that, especially if you use the famous package-manager `use-package`, you might need to start it more than once to get every package installed.