Instruct

Author: Yuri Tricys
Date: December 19th, 2025

PIDP:3270 CAPSTONE PROJECT

OPEN SOURCE SOFTWARE DEVELOPMENT: BUILDING EMACS FROM SOURCE: LESSON PLAN

VCC Build Hello

Introduction

This report provides a lesson plan for an 8-part video series on building the Emacs text editor (version 30.2) from source, primarily on Ubuntu 25.04 and secondarily on WSL (Windows 11 with WSLg) and macOS. The lesson walks learners from "Why Emacs?" through downloading, verifying, configuring, and building Emacs — culminating in a complete source build inside a Docker container.

The plan follows the BOPPPS structure (Bridge-in, Objectives, Pre-assessment, Presentation, Practice, Summary) and is detailed enough that a substitute instructor can deliver the lesson using the provided videos and transcripts.

The Lesson Plan

Lesson Metadata

Course	Mastering Emacs for Creatives
Lesson	Building Emacs 30.2 from Source (Linux, WSL, and macOS)
Time	~2 hours of video + 1–2 hours of hands-on practice (8 videos, ~10–20 minutes each)

Resources Required	• Browser to watch the videos.
	• Computer with one of:
	  ◦ Ubuntu 25.04 (native or in a VM), or
	  ◦ Windows 11 with WSL2 + WSLg, or
	  ◦ macOS with Homebrew (for parallel practice; examples are Linux-centric).
	• `gpg`, and at least one of `wget`, `curl`, or `aria2c` to retrieve the source files.
	• `tar` (available on Linux, macOS, and in most shells, including PowerShell).
	• Package manager:
	  ◦ Ubuntu/Debian: `apt`
	  ◦ Fedora/RHEL: `dnf`
	  ◦ Arch: `pacman`
	  ◦ openSUSE: `zypper`
	  ◦ macOS: `brew`
	  ◦ Windows: `winget`, `choco`, or `scoop` (for prebuilt binaries).
	• `docker` (optional but used in Video 8; recommended for learners who want a clean, reproducible build environment).
	• Any text editor (Notepad, nano, Emacs, VS Code, etc.) to keep a build log and notes.

Resources Provided	• URLs to all 8 lesson videos. • Downloadable transcripts for each video. • A “quick recipe” document with copy-paste terminal commands for: ◦ Downloading and verifying Emacs 30.2 source. ◦ Installing build dependencies on Ubuntu 25.04. ◦ Running `configure`, `make`, and `make install`. • Links to authoritative documentation: ◦ Official GNU Emacs download and build pages. ◦ Ubuntu / WSL / Docker documentation. ◦ GNU Autotools overview articles. • A downloadable repository (or archive) with: ◦ Example build scripts (for minimal and “full” builds). ◦ Sample Dockerfile used in Video 8. ◦ Example Emacs Lisp helpers to inspect features and configure options.

Lesson Bridge In

Instructor Activities	Time
Instructor motivates why Emacs matters, and why building from source is worth learning.	5 minutes

You open your laptop to write, code, or analyze data – and most of your time goes into managing windows and tools rather than doing the creative work itself. Emacs exists to flip that balance.

Explain to learners:

- Emacs is more than a text editor:
  - It’s an IDE, a note-taking system, an email client, a reproducible research environment, and an AI front-end.
  - It’s keyboard-centric: “hands never leave the keyboard,” so context switching drops dramatically.
- Emacs can 10x editor-related productivity:
  - Navigating large projects, organizing notes, managing email and tasks.
  - Interacting with AI agents directly inside an editor buffer.
- You can get Emacs quickly from a package manager – but building from source:
  - Improves trust (you verify exactly what you’re running).
  - Lets you customize features and performance for your machine.
  - Teaches you how open-source software is really built.

Set the context for this 8-video lesson:

- We'll:
  - Decide when to use a package manager vs. build from source.
  - Download and verify Emacs 30.2 source from the GNU servers.
  - Learn what GNU Autotools do (autoconf, configure, make, GCC).
  - Install the key build dependencies for a modern GUI Emacs.
  - Choose and understand important `configure` options.
  - Build Emacs – first in a normal Ubuntu environment, then reproducibly inside a Docker container.

Clarify platforms:

- Demonstrations are done on Ubuntu 25.04.
- WSL users can follow the exact same shell commands inside a WSL Ubuntu instance (with WSLg for GUI).
- macOS users can adapt the package installation steps (Homebrew) and focus on the same conceptual flow.

Close the bridge by emphasizing:

- By the end of this lesson sequence, learners will not only have a custom Emacs build installed – they'll understand what happens when open-source software "builds itself" on their machine.

Lesson Objectives

Instructor Activities	Time
Instructor clearly states learning objectives and briefly maps each objective to the corresponding video(s).	3 minutes

- Conceptual Objectives
  - Explain at least three reasons why Emacs is valuable for writers, developers, researchers, and other knowledge workers. (Videos 1–2)
  - Compare the trade-offs between installing Emacs via a package manager and building it from source (Video 3)
  - Describe, at a high level, the roles of `autoconf` / `configure`, `make`, and `gcc` in a typical GNU Autotools-based build. (Video 5)
- Technical Objectives
  - Install required command-line tools (`gpg`, `wget` or `curl` or `aria2c`, `tar`) on Ubuntu / WSL / macOS. (Video 4)
  - Download the Emacs 30.2 source tarball and detached signature from the official GNU server and verify integrity with GPG. (Video 4)
  - Extract the Emacs source tree using `tar`. (Video 4)
  - Install a minimal but modern set of build dependencies (GTK GUI, image libraries, TLS, SQLite, Tree sitter, native compilation) on Ubuntu 25.04. (Video 6)
  - Run `./configure` with a provided set of options to:

        - Select the Wayland/PGTK GUI backend.
        - Enable native compilation and Tree-sitter.
        - Set a custom `--prefix` when appropriate.
        - Apply safe compiler optimization flags. (Video 7)
    - Build and install Emacs with `make -jN` and `sudo make install`, and confirm that the resulting binary runs. (Video 8)
    - Optionally reproduce the build inside a Docker container based on Ubuntu 25.04. (Video 8)
- Meta-Skills Objectives
    - Document build steps, errors, and fixes in a personal log for future reproducibility.
    - Use Emacs Lisp helper functions to inspect:
        - Build features (`system-configuration-features`).
        - Configure options compiled into a binary. (Video 7)

Lesson Pre-Assessment

Given this is an asynchronous lesson, pre-assessment is handled as a self-check and gentle discussion of prerequisites at the start of the series.

Instructor Activities	Time
Instructor outlines technical prerequisites and asks learners to self-assess their readiness and environment.	5 minutes

Points for the instructor to cover (spoken, with slides):

1. **System readiness**
    - Do learners:
        - Have an Ubuntu 25.04 system (native or WSL2) or are they comfortable adapting to another distro?
        - Have 2–4 GB free disk space and reasonable CPU/RAM (Emacs itself is small, but dependencies and Docker images can be larger)?
        - Have administrative rights (can use `sudo` or equivalent)?
2. **Shell and OS familiarity**
    - Learners should:
        - Know how to open a terminal (GNOME Terminal, Windows Terminal, iTerm2).
        - Be comfortable running basic commands: `cd`, `ls`, `mkdir`, `sudo apt install`.
        - Understand that paths and package managers differ slightly between Linux distros and macOS.
3. **Tools to install before starting Video 4**
    - On Ubuntu / WSL (example):

        -
          ```
          sudo apt update
          sudo apt install -y gnupg wget curl aria2 tar
          ```
    - On macOS:
        - `brew install gnupg wget curl aria2`

4. **Learning mindset and note-taking**
   - Ask learners to open a small “build log” file (any editor is fine). Suggest they record:
     - Commands they run (copy/paste from video or slides).
     - Errors they encounter (verbatim error messages).
     - What they tried (including prompts they may feed to AI).
     - What ultimately fixed the issue.
5. **Pre-lesson self-check prompts (optional)**
   - Ask learners to pause and answer for themselves:
     1. Have I used a terminal before? If not, am I willing to take it slowly and copy commands carefully?
     2. Can I install software with my system’s package manager (e.g., `apt`, `dnf`, `brew`)?
     3. Do I feel comfortable reading short error messages and trying suggested fixes?

Encourage learners that no deep C or Unix internals knowledge is required; willingness to follow steps carefully and debug simple issues is enough.

Lesson Participatory Learning: Video 1 & 2

Instructor Activities

Instructor Activities	Time
Video 1–2: Introduce the course arc and the “why Emacs” motivation (unified workspace, longevity, learning curve expectations, cross-platform applicability, and suggested external resources).	~20–25 minutes

Instructor activity details (spoken, with slides/demo):

1. **Video 1 — Introduction to Building Emacs From Source**
   - Emacs as a “productivity robot” / long-lived open-source editor.
   - Course arc: why Emacs → how to get it → building from source → later configuration and reproducible research.
   - Set expectations: Emacs mastery takes time; this course accelerates the ramp.
   - Cross-platform applicability: Linux, WSLg on Windows 11, macOS.
2. **Video 2 — Reasons To Use Emacs**
   - Emacs as IDE + unified workspace:
     - Use cases: coding, docs, data analysis, web dev, note-taking, email, AI interaction.
     - Users: developers, writers, data scientists, academics, musicians, and non-tech creatives.
   - Popularity slide; emphasize customizability and longevity.
   - “Time travel for your workflow”: fewer apps, more keyboard workflows, less context switching.
   - “10x productivity” framing (after basics are mastered).
   - External resources:
     - System Crafters (recommend a specific intro video).

- Emacs conferences; Emacs blogs (e.g., Reading World Magazine).
- Reassurance: it's okay if Emacs feels overwhelming; the series narrows the path deliberately.

Learner Activities

Learner Activities	Time
Watch Videos 1 and 2.	~20–30 minutes
In notes, list: (a) 3 tasks you currently do in other tools that Emacs might unify; (b) 1–2 concerns you have about Emacs's learning curve.	~5–10 minutes
Optional: Watch the recommended System Crafters intro video and note keybindings/concepts (e.g., `M-x`, `C-x C-f`).	~15–25 minutes

Learner activity details (what to produce):

1. *Video viewing*
   - Watch Videos 1 and 2 straight through (pause to jot quick notes).
   - Goal: understand the course arc and why building from source matters later.
2. *Notes deliverable (your "why Emacs" baseline)*
   - Write 3 concrete tasks you do today (examples: coding + git, note-taking, writing docs, project planning, email, terminal workflows).
   - Write 1–2 learning-curve concerns (examples: keybindings, configuration complexity, plugin management).
3. *Optional external primer (System Crafters)*
   - Capture 3–5 items: a keybinding, a concept (buffers/windows), and one workflow idea you want to try later.
   - Save these notes—you'll revisit them when configuring your own Emacs.

Lesson Participatory Learning: Video 3

Instructor Activies

Instructor Activities	Time
Video 3: Compare installing Emacs via a package manager vs building from source; explain tradeoffs, show representative install commands across platforms, and set expectations for what this course will emphasize.	~15 minutes

Instructor activity details (spoken, with slides):

1. **Why package manager first (for most users)**
   - Simplicity, dependency handling, security updates, clean uninstall.
2. **Why build from source anyway (for this course)**
   - Security/integrity (GPG verification).
   - Customization (configure flags; minimal vs full).

- Optimization (CPU-specific flags).
- Latest features (stable tarball vs Git builds).
- Non-standard prefixes; multiple versions side-by-side.
- Reproducibility/containers; sharing environments.
- Debugging options during configure/make.
- Education: learning an Autotools-style build.

3. **Concrete package-manager commands (examples)**
   - Windows: `choco install emacs`, `winget install Emacs.Emacs`, `scoop install emacs`; MSYS2/Cygwin notes.
   - macOS: `brew install --cask emacs`; official GNU Emacs macOS builds.
   - Linux: `sudo apt install emacs`, `sudo dnf install emacs`, `sudo pacman -S emacs`, `sudo zypper install emacs`, etc.
   - Mention Flatpak/Snap sandboxing caveats for Emacs integrations.

Learner's Activies

Learner Activities	Time
Watch Video 3.	~10–20 minutes
Decide your approach: keep a package-manager Emacs as baseline, or rely primarily on a source build.	~5–10 minutes

Learner activity details (what to produce):

1. *Comparison notes*
   - Make two columns and write at least 3 advantages for each approach.
   - Keep the advantages specific (e.g., "security updates via distro" vs "newer features sooner").
2. *Decision + rationale*
   - Choose one primary path (baseline + source build is often best).
   - Write a 1–2 sentence rationale so you remember why you chose it later.

Lesson Participatory Learning: Video 4

Instructor Activities

Instructor Activities	Time
Video 4: Demonstrate downloading GNU Emacs source and signature files, obtaining the GNU keyring, verifying the tarball with GPG, and extracting the source tree.	~25–30 minutes (video + doing the steps)

Instructor activity details (spoken, with live terminal demo):

1. **Explain GPG's role**
   - GnuPG/OpenPGP basics; authenticity + tamper detection for tarballs.

2. **Install required tools (platform examples)**
   - Ubuntu: `sudo apt update`; `sudo apt install -y gnupg wget curl aria2`
   - Windows (Chocolatey): `choco install gpg4win wget curl aria2`
   - macOS (Homebrew): `brew install gnupg wget curl aria2`
   - Verify: `gpg --version`
3. **Download Emacs release artifacts**
   - Locate Emacs 30.2 `.tar.gz` and `.sig` on official mirror pages.
   - In terminal/eshell:
     - Create source dir (e.g., `~\/local\/src\/emacs` or `~\/src\/emacs`).
     - Download with `wget~/~curl~/~aria2c`:
       - `emacs-30.2.tar.gz`
       - `emacs-30.2.tar.gz.sig`
4. **Download GNU keyring**
   - Obtain `gnu-keyring.gpg`; explain maintainer keys (e.g., Eli Zaretskii) live there.
5. **Verify and interpret output**
   - Run:
     - `gpg --keyring ./gnu-keyring.gpg --verify emacs-30.2.tar.gz.sig emacs-30.2.tar.gz`
   - Point out "Good signature …"; explain "not certified with a trusted signature" warning.
6. **Extract and orient to the tree**
   - `tar xzf emacs-30.2.tar.gz` (mention `tar xJf` for `.tar.xz`)
   - Show top-level layout: `README`, `src/`, `lisp/`, `configure`, etc.

Learner Activities

Learner Activities	**Time**
Follow along in your terminal: install GPG/download tools (if needed), create a source dir, download `emacs-30.2.tar.gz` and `.sig`, download `gnu-keyring.gpg`, run `gpg --verify`, extract with `tar xzf`, and confirm contents with `ls`.	~25–30 minutes
Maintain a build log capturing exact commands, verification output, and any errors/resolutions.	~5–10 minutes

Learner activity details (what to produce):

1. *Verified source download*
   - Ensure you can reproduce the download + signature verification.
   - Record the "Good signature" line and the signer/maintainer name.
2. *Build log artifact*
   - Keep a plain-text log (or Org file) that contains:
     - Commands run (copy/paste).
     - GPG output.

- Any troubleshooting steps you took.

Lesson Participatory Learning: Video 5

Instructor Activities

Instructor Activities	Time
Video 5: Explain GNU Autotools and the standard build pipeline (configure → make → make install), using a metaphor and pointing students to relevant docs.	~20 minutes

Instructor activity details (spoken, with slides):

1. **Autotools metaphor**
   - Emacs as a robot building itself on Mars.
   - `configure` inventories parts; `make` orchestrates assembly.
2. **Conceptual breakdown**
   - autoconf: generates `configure` (mostly for Git builds).
   - configure:
     - Checks compilers/headers/libs; decides enabled features.
     - Writes `Makefile` and config headers.
   - make / Makefile:
     - Dependency-driven compilation; `gcc` flags.
   - make install:
     - Copies binaries/lisp/docs to prefix.
     - Discuss default prefix (`/usr/local`) and avoiding overwriting distro Emacs.
3. **Common pipeline**
   - Optional: `./autogen.sh` (Git builds)
   - `./configure <options>`
   - `make -jN`
   - `sudo make install`
4. **Where docs live**
   - Point to `INSTALL` and other build-system docs.

Learner Activities

Learner Activities	Time
Watch Video 5.	~10–20 minutes
Create a 4-box pipeline in notes: `configure` → `make` → `make install` → run `emacs`, with a 1-sentence description under each.	~5–10 minutes

Learner activity details (what to produce):

1. *Conceptual build model*
   - Write what each stage does in your own words (one sentence each).
   - This is your mental checklist for debugging later.
2. *Prior experience reflection (optional but helpful)*
   - Note anything familiar (Autotools, Makefiles) and anything new.

Lesson Participatory Learning: Video 6

Instructor Activities

Instructor Activities	Time
Video 6: Present Emacs build dependencies for a “full” modern build, explain what each category enables, and demonstrate installing them (Ubuntu-focused, with notes for other distros).	~20 minutes (video + installation)

Instructor activity details (spoken, with slides + a single install demo):

1. **Reinforce dependencies metaphor**
   - Dependencies as “parts” the robot finds; features depend on what’s available.
2. **Curated dependency set (Ubuntu 25.04 target)**
   - Build tools: `build-essential`, `pkg-config`, plus Git-build helpers (`autoconf`, `automake`, `autopoint`, `libtool`).
   - Core runtime/terminal/compression: `zlib1g-dev`, `libncurses-dev`, `libsqlite3-dev`, `libgmp-dev`, `libgpm-dev`.
   - TLS: `libgnutls28-dev` (or equivalent).
   - GUI/rendering: `libgtk-3-dev`, `libxpm-dev`.
   - Image formats: `libgif-dev`, `librsvg2-dev`, `libwebp-dev`, `libjpeg-dev` / `libjpeg-turbo-dev`, `libtiff-dev`, `libavif-dev` (optional), `liblcms2-dev`.
   - Text/internationalization: `libharfbuzz-dev`, `libxml2-dev`, optional `libotf-dev`, `libm17n-dev` (X11 multi-lingual; optional for PGTK).
   - Native comp & tree-sitter: `libgccjit-14-dev` (match system gcc), `libtree-sitter-dev`.
   - Mail utils: `mailutils` (mostly for configure checks).
   - Fedora/SELinux concept mention: `libacl`, `libselinux` headers.
3. **Why “full build” now**
   - Not strictly required for minimal Emacs; reduces later friction.
4. **Demo install + prompts**
   - Run one `sudo apt install -y <long list>`.
   - Walk through `mailutils` prompts (timezone/email host/dummy domain).
5. **Advanced note**
   - Brief mention: AppArmor/SELinux can interfere; out of scope.

Learner Activities

Learner Activities	Time
Watch Video 6.	~20 minutes
Install required build dependencies (Ubuntu/WSL: run the provided `sudo apt install -y ...`; other distros: adapt via `dnf` / `pacman` / `zypper`).	~5–10 minutes
Document which packages installed, which were missing, and run `gcc --version` (record output).	~5–10 minutes

Learner activity details (what to produce):

1. *Dependency install record*
   - Save the final list of installed packages (or the command that installed them).
   - Write down any package-name differences on your distro.
2. *Compiler version record*
   - Capture the `gcc --version` output for later alignment with `libgccjit` expectations.

Lesson Participatory Learning: Video 7

Instructor Activities

Instructor Activities	Time
Video 7: Teach how to choose and understand Emacs configure options (X11/GTK vs Wayland/PGTK), including prefix strategy, feature flags, and performance-related CFLAGS; then demonstrate running ./configure and reading the summary.	~20 minutes (video + running configure)

Instructor activity details (spoken, with slides + command-line demo):

1. **Frame configure as "blueprint generator"**
   - Explain that the summary output reflects detected libraries/features.
2. **Inspecting an existing Emacs build (optional helpers)**
   - Emacs Lisp helpers to copy `system-configuration-features` and inspect the original distro/flatpak configure line.
3. **Two templates**
   - X11/GTK build (older/X11-first environments).
   - Wayland/PGTK build (recommended for modern systems):
     - Use `--with-pgtk` to force PGTK; explain GTK3 defaults vs explicit PGTK.
4. **Key options used in the course full build**
   - `--prefix=...` (avoid overwriting system Emacs; allow multiple versions).
   - `--with-pgtk`
   - `--with-native-compilation`
   - `--with-tree-sitter`, `--with-json`, `--with-modules` (as included)
   - `--disable-silent-rules` (verbosity for learning/debugging)

   - `--disable-gc-mark-trace` / `--disable-gc-malloc-debugging` (performance vs debug tradeoff)

5. **CFLAGS discussion**
   - `-O2` vs `-O3` vs `-Os`
   - `-march=native` vs portability concerns
   - `-mtune=native`, `-fno-math-errno`, `-g`, `-fomit-frame-pointer`
   - Mention `-fsanitize=undefined` as debug-only.
6. **Show a final sample configure command**
   - Provide the multi-line `./configure ... CFLAGS="..."` example and point to official ~INSTALL~/configure docs.

Learner Activities

Learner Activities	Time
Watch Video 7.	~20 minutes
Optional: In an existing Emacs, run the helper to inspect features/past configure options.	~5–10 minutes
Choose a GUI backend (Wayland/PGTK if available, otherwise X11/GTK) and copy/adapt the instructor sample `./configure` command (choose `--prefix`, keep recommended `CFLAGS` unless you know why you're changing them).	~5–10 minutes
Run `./configure` inside `emacs-30.2/` and save the tail "summary" output in your build log.	~5–10 minutes

Learner activity details (what to produce):

1. *Decision record (backend + prefix)*
   - Write down which backend you chose and why (one sentence).
   - Write down your chosen install prefix and where Emacs will end up on disk.
2. *Configure summary*
   - Save the configure command you used.
   - Paste the configure summary (end of output) into your log so you can compare later.

Lesson Participatory Learning: Video 8

Instructor Activities

Instructor Activities	Time
Video 8: Demonstrate a reproducible Emacs build inside Docker (including display sharing considerations), then configure, compile, test-run, and optionally install Emacs; explain how to restart/exit and how to compare minimal vs full builds.	~45 minutes (video + full build)

Instructor activity details (spoken, with Docker + terminal demo):

1. **Why Docker here**
    - Clean Ubuntu 25.04 environment; reproducibility; avoid disturbing host OS.
    - Useful for comparing minimal vs full feature builds.
2. **Docker install (high-level)**
    - Remove old packages; add Docker repo + GPG key; install `docker-ce...`
    - Start and check service: `systemctl start/status docker`.
3. **Launch container (show example; note environment-specific adaptation)**
    - Demonstrate `docker run ...` with Wayland/X11 display sharing as appropriate.
4. **Inside container setup**
    - Create non-root work dir (e.g., `/home/ubuntu/src`).
    - Configure apt sources via heredoc into `ubuntu.sources`.
    - `apt update`; install core tools + same dependencies as Video 6.
    - Handle `mailutils` prompts.
5. **Repeat download/verify/extract in container**
    - Re-run Video 4 steps inside the container.
6. **Minimal vs full build concept**
    - Show how configure summary changes with fewer/more deps.
7. **Configure → build → test**
    - Run `./configure ...` and interpret summary (PGTK vs X11, image libs, TLS, HarfBuzz, tree-sitter, native comp).
    - Determine cores: `nproc`
    - Build: `make -j$(nproc)`
    - Test without install: run `src/emacs`; optionally inspect `system-configuration-features`.
8. **Install + wrap up**
    - `sudo make install` (system prefix) or `make install` (user prefix).
    - Exiting/restarting container (`exit`, `docker start -ai ...`).
    - Note copying artifacts to host (conceptual).

Learner Activities

Learner Activities	Time
Watch Video 8.	~45 minutes
Decide whether to build directly on Ubuntu/WSL or use Docker as an exercise (then install and configure Docker).	~5–10 minutes
Run the build steps: `./configure`, `make -j$(nproc)`, `sudo make install` (or non-sudo with user prefix).	~5–10 minutes
Test the result: run `emacs -Q` and confirm GUI launch if applicable.	~5–10 minutes

Learner Activities	Time
Update build log with configure line, CPU core count, approximate build time, and any errors/resolutions.	~5–10 minutes

Learner activity details (what to produce):

1. *Reproducible build record*
   - Record: build location, configure line, make command, install command.
   - Record: number of cores used and rough build duration.
2. *Validation*
   - Confirm `emacs -Q` starts (this isolates config issues).
   - Note whether you launched terminal Emacs, GUI Emacs, or both.
3. *Troubleshooting notes*
   - If anything fails, paste the key error lines and what fixed it (or what you tried).

Lesson Post-Assessment

Since this is an asynchronous lesson, use reflective prompts and a short self-check instead of a formal test. Encourage learners to pause, write answers in their notes, and, if possible, discuss with peers or in an online forum.

Instructor Activities	Time
Instructor poses reflection and self-assessment questions to consolidate learning.	5–10 minutes

Suggested prompts:

1. **Expectations vs reality**
   - When you started, how did you imagine “building Emacs from source” would feel?
   - Now that you’ve done it (or attempted it), what was easier than expected? What was harder?
2. **Understanding the build process**
   - In your own words, describe:
     - The purpose of `configure`.
     - The purpose of `make`.
     - The purpose of `make install`.
   - How does this pipeline generalize to other open-source software you might build in the future?
3. **Debugging mindset**
   - Name one error you encountered (or could plausibly encounter), and outline your debugging steps.
     - Example: Missing library reported by `configure`.
       - Check package manager for that library.
       - Install it.
       - Re-run `./configure`.

- If stuck, search the exact error message or ask an AI with the error text.

4. **Why build from source?**
    - List two reasons someone might build Emacs from source even if a package manager version is available.
    - Which of those reasons are most important to you personally right now?
5. **Feature awareness**
    - Open your newly built Emacs and inspect `system-configuration-features`.
    - Can you find evidence that:
        - PGTK or X11 backend is enabled?
        - Native compilation is enabled?
        - TLS and image support are active?

Encourage learners to add a “Lessons learned” section at the end of their build log summarizing the main takeaways.

Lesson Summary

Instructor Activities	Time
Instructor summarizes key achievements and connects this build lesson to upcoming Emacs customization and reproducible research topics.	5 minutes

Key points for the wrap-up:

- Learners have:
    - Seen concrete reasons to adopt Emacs for serious text and knowledge work.
    - Compared installation via package managers to building from source.
    - Downloaded and cryptographically verified the Emacs 30.2 source code.
    - Installed a modern dependency set for a feature-rich GUI Emacs.
    - Understood, at a practical level, how Autotools, `configure`, `make`, and `gcc` collaborate to build software.
    - Built and run Emacs from source, either on their host system or in a Docker container.

Pose a final checklist for self-evaluation:

- Can you explain at a high level how open-source software like Emacs is built from source?
- Can you reproduce a working Emacs build on a new Ubuntu/WSL machine using your own notes?
- Do you know how to:
    - Verify source tarballs with GPG?
    - Install and align toolchains (e.g., `gcc` and `libgccjit`)?
    - Interpret the `configure` summary and adjust dependencies accordingly?
- Do you feel more confident approaching other Autotools-based projects from source?

Conclude by previewing the next phase of the course:

- Now that Emacs is built and installed, the next lessons will:
    - Show where your Emacs configuration lives (`early-init.el`, `init.el`, configuration directories).
    - Introduce a reproducible research pipeline built inside Emacs (e.g., Org-mode-driven configuration).
    - Provide a complete starter configuration and gradually teach you how to extend and understand it.

Encourage learners to keep their custom build notes handy — they are both documentation and the start of their own reproducible research practice.

Media Examples

This course can be delivered in several formats using the same 8-part content structure:

- Synchronous, remote (e.g., Zoom):
    - Instructor shares screen: slides + terminal + Emacs.
    - Live demos mirror what happens in the recorded 8-part series.
    - Students follow along on their own machines; instructor pauses regularly for questions and troubleshooting.
- Synchronous, in-person (computer lab or classroom):
    - Projector shows instructor's screen.
    - Each student has a workstation with a shell and a web browser.
    - Instructor alternates between:
        - short explanations (slides/whiteboard), and
        - short build steps they demonstrate in a terminal/Emacs.
    - Students reproduce each step locally before moving on.
- Asynchronous (self-paced video lesson):
    - Students watch the 8-part video series on their own schedule.
    - Each video is paired with:
        - a short reading/cheat sheet (commands and URLs), and
        - a small progress task (e.g., "By the end of Part 3, you should have downloaded and verified the Emacs tarball.").
    - Optional: weekly office hours (in-person or online) for Q&A and debugging help.

An instructors can choose whichever mode matches the schedule and facilities, but the *sequence of concepts and tasks* should follow the same 8-part outline.

OBS Setup → Remote Teaching Setup

If teaching **live online**, use any video-conferencing tool (Zoom, Meet, Teams) with simple screen-sharing instead of OBS:

- Recommended screen-sharing layout:
    - One desktop with:

        - A terminal window (for shell commands: `wget`, `gpg`, `configure`, `make`, etc.).
        - An Emacs window (for editing config files, exploring source, etc.).
        - Optional: a slide deck (for key diagrams and bullet points).
    - Share:
        - The *entire desktop* if you switch among terminal/Emacs/slides, or
        - A *single window* (terminal/Emacs) if you want to minimize distractions.
- Use of the instructor's camera:
    - Camera on during introductions, explanations, and Q&A can help engagement.
    - For bandwidth-limited students, prioritize a clear screen share over video.
- Interaction patterns:
    - Encourage students to:
        - type commands along with you,
        - paste error messages in chat,
        - use "raise hand" or reactions when they get stuck.
    - Pause after each major step (e.g., "Download done?", "Configure finished?", "Any build errors?").

No special OBS features are required; a stable screen share and clear audio are sufficient.

Example of OBS Usage → Example Teaching Flow (Remote or In-Person)

Instead of focusing on OBS techniques, here's how to *run one of the 8 parts* in a live setting:

1. **Bridge-in (2–3 minutes)**
    - Explain what this segment will cover.
    - Example (for the "configure & build" segment):
        - "In this part, we'll run the `configure` script, talk about build options, and compile Emacs from source on your machine."
2. **Demonstration (10–15 minutes)**
    - Show your terminal:
        - Run the commands slowly, narrating each step:
            - e.g., "Now I run `./configure --with-x-toolkit=gtk3` to enable the GTK GUI. Your options may differ based on your system."
    - Switch briefly to slides (if available) to:
        - explain what `configure` checks for,
        - show a diagram of the build pipeline: source → configure → make → install.
3. **Guided Student Practice (10–20 minutes)**
    - Have students run the same commands:
        - Ask them to say when they get errors.
        - Show them how to read error messages and search logs.
    - Collect a few common errors, share screen, and walk through resolving them.
4. **Wrap-up (3–5 minutes)**
    - Summarize:

    - "You should now have a successful `make` build in the `src` directory."
  - Preview the next segment:
    - "Next time we'll run our newly built Emacs, customize where it's installed, and verify which binary you're running."

This pattern (bridge-in → demo → practice → wrap-up) can be reused for all 8 parts, whether online or in-person.

Media and Resources Required

To deliver the course in the different modes:

For Synchronous Remote (Zoom or similar)

- A computer with:
  - Internet access.
  - A terminal (Linux/macOS or WSL on Windows).
  - Emacs installed (even if the goal is to rebuild it from source).
- Video-conferencing platform (Zoom/Meet/Teams) with:
  - Screen sharing.
  - Chat and/or "raise hand" for questions.
- Optional:
  - A copy of the 8-part videos to reference or assign as pre-work.

For In-Person (Computer Lab / Classroom)

- Room setup:
  - Projector or large display for instructor's screen.
  - Whiteboard or flipchart for diagrams (e.g., build pipeline).
- Student workstations:
  - Each system should have:
    - A Unix-like environment (Linux, macOS, or WSL).
    - Internet access to download Emacs source and build dependencies.
  - Preferably identical or similar OS/distributions to reduce environment-specific issues.
- Printed or digital handouts:
  - Command reference for each part (copy-paste friendly).
  - URLs for Emacs source, documentation, and reference materials.

For Asynchronous Delivery

- Hosting for:
  - The 8 videos (e.g., LMS, private YouTube, institutional server).
  - Supporting files:
    - Text versions of commands used in each video.
    - Short quizzes or checklists (e.g., "Can you locate your built Emacs binary?").

- Communication channel for help:
    - Discussion forum, chat (Slack/Discord), or scheduled office hours.
- Clear instructions:
    - Estimated time per video.
    - What students should *do* after each part (e.g., “By the end of Part 8 you should have run your locally built Emacs and confirmed its version.”).

Conclusion

This report outlined a detailed lesson plan, for teaching learners how to build Emacs 30.2 from source – primarily on Ubuntu 25.04 and WSL, with concepts adaptable to macOS and other Linux distributions.

The plan:

- Follows BOPPPS to give structure: motivation, objectives, pre-assessment, guided practice, and reflection.
- Makes explicit:
    - Why Emacs is worth learning.
    - When to use package managers vs. source builds.
    - How to download, verify, configure, and compile Emacs.
    - How to reason about dependencies, build toolchains, and configure options.

An instructor can follow this plan, use the existing videos and transcripts, and reliably guide students through the process of building Emacs from source – laying the foundation for subsequent lessons on Emacs configuration, reproducible research workflows, and advanced productivity techniques inside Emacs.

References

[Photograph of a rocket] [Photograph]. (n.d.). Internet Archive. Retrieved December 10, 2025, from https://archive.org/ (https://archive.org/)

AI Models Used In This Report: Various AI models were used to research, collect, and verify data, format arguments, and grammatically structure content. Models used include: Qwen3-coder:30b-a3b-q4_K_M, Gemma3:27b-it-q4_K_M, Llama3.1:70b-instruct-q3_K_M, GPT 5 Nano, GPT 5.1