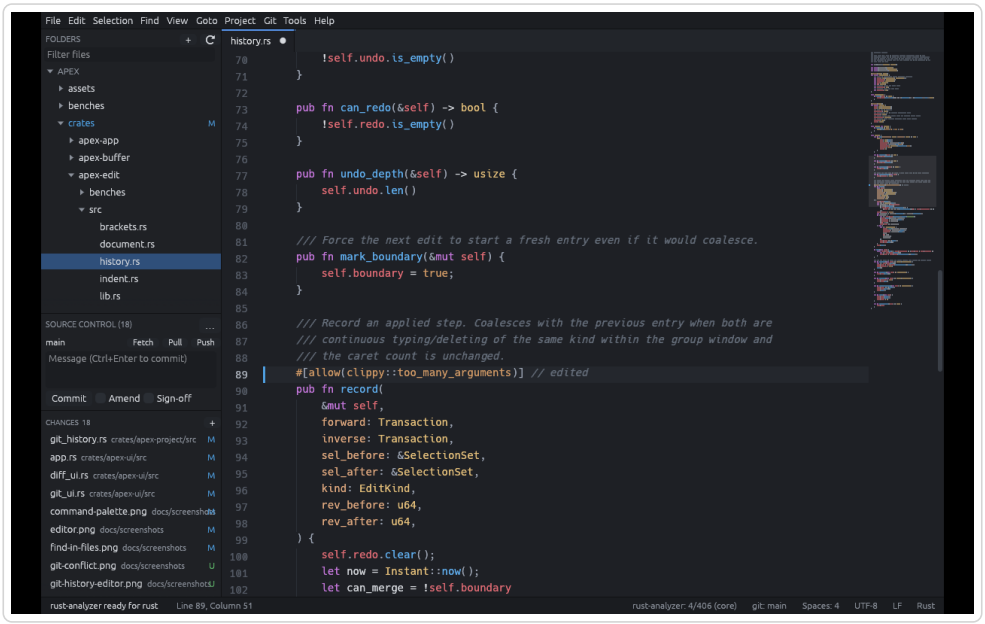
apex.

The User Guide

A fast, native text editor.
No plugins, no accounts, no telemetry.



Apex starts instantly, edits with many cursors at once, finds anything in a project in a keystroke, and understands Git. Everything works offline. **Read in any order:** every chapter opens with a **TL;DR** and ends with a **Done when** check, so you always know when you can stop.

How to use this guide

TL;DR

You do not need to read this in order. Every chapter starts with a box like this one that says what the chapter is for. Read the box; if it is what you need, carry on; if not, skip the chapter.

The key, so you can scan

Marker	Means
`Ctrl+P`	A key to press. `Ctrl+K Ctrl+S` is a chord: one, then the other.
File > Open Folder	A menu path, or a button to click.
`settings.toml`	Something to type, or a file name.
TL;DR	The chapter in a few lines. Read only this if you are in a hurry.
	Roughly how long the chapter takes.
Try it	A two-minute exercise. Doing one beats reading a page.
Watch out	The things that catch people out.
Your data	What Apex does, and does not do, with your files and credentials.
Done when	How you know you have finished the chapter.

Shortcuts are written for Linux and Windows. On macOS read `Ctrl` as `Cmd` unless the text says otherwise. **Tools > Keyboard Shortcuts** (`Ctrl+K Ctrl+S`) lists every binding for your platform, and the command palette shows the shortcut next to each command.

The one-page cheat sheet

Keys	What happens
`Ctrl+P`	Goto Anything: files, `:line`, `@symbol`, `#project symbol`, `>command`
`Ctrl+Shift+P`	Command palette: every command, with its shortcut
`Ctrl+O` / `Ctrl+Shift+O`	Open a file / open a folder
`Ctrl+S`	Save (atomic: a half-written file never replaces yours)
`Ctrl+D`	Select the word, then add the next occurrence each press
`Ctrl+Shift+L`	One cursor per selected line
`Ctrl+F` / `Ctrl+H`	Find / replace in the file
`Ctrl+Shift+F`	Find in files across the project
`Ctrl+B`	Run the build task
`Ctrl+K Ctrl+G`	Git blame on or off
`Ctrl+\`	Split the pane
`F11` / `Shift+F11`	Fullscreen / distraction-free
`Escape`	Back to one cursor, close the bar, leave the mode

Contents

How to use this guide 1

1 Getting started 6

Installing 6

The window 6

Opening things 7

The command line 7

Where things live 7

Updates 8

2 Editing 9

Everyday keys 9

Multiple cursors 10

Bookmarks 11

Snippets 11

Macros 12

Spell checking 12

The hex editor 12

3 Navigation 13

Goto Anything (`Ctrl+P`) 13

Command palette (`Ctrl+Shift+P`) 14

Moving around 14

Panes and tabs 15

4 Search and replace 16

In the file 16

Across the project 17

5 Projects, the sidebar and sessions 18

Folders 18
Remote files over SSH 18
Project files 19
Sessions and recovery 20

6 Configuration 21

Settings 21
Key bindings 22
Themes 22
Dictionaries 23

7 Git 24

What you see without doing anything 24
The Source Control panel 25
Side-by-side diffs 26
Staging 26
Fetch, pull and push 26
GitHub and other code hosts 26
Branches, log and stash 27
Merge conflicts 28
Blame 28
Edit Commit History 29
The GIT output tab 30

8 Build tasks 31

Where tasks come from 31
Trust 31
Problems 32

9 Language servers 33

Starting a server 33
What you get 34

10 Vim mode 35

Modes and motions 35
Operators and text objects 36
Registers, repeat, command line 36

11 Troubleshooting 37

CHAPTER 1

Getting started

TL;DR

Install Apex, open a folder, and learn the shape of the window. Ten minutes.

Ten minutes

1.1 Installing

Linux
: Unpack `apex-<version>-linux-<arch>.tar.gz` and run `./install.sh` (installs to `~/.local`; prefix with `PREFIX=/usr/local sudo` for all users). Or install the `.deb` with `sudo apt install ./apex_<version>_amd64.deb`. The AppImage runs from anywhere after `chmod +x`.

macOS
: Open the DMG and drag `Apex.app` to Applications. Apex appears in Finder's **Open With** menu for any file.

Windows
: Run `Apex-<version>-setup.exe`. The installer can add **Open with Apex** to the Explorer menu, associate common text files and put `apex` on the PATH. The zip is a portable alternative.

Building from source is described in the project's `README.md`.

1.2 The window

```
Menu bar
+-----------+-----------------------------------------+
| Sidebar   | Tabs                                    |
| Folders   +-----------------------------------+-------+
| (project  | Editor                            |Minimap|
|  files)   |                                   |       |
+-----------+-----------------------------------+-------+
| Output / Problems / Find results / GIT panel        |
+-----------------------------------------------------+
| Status bar: position, git branch, language, encoding |
+-----------------------------------------------------+
```

Everything around the editor can be hidden from the **View** menu (`Ctrl+K Ctrl+B` toggles the sidebar). `F11` is fullscreen, `Shift+F11` distraction-free, `Ctrl+K Ctrl+Z` zen mode; `Escape` leaves all three.

1.3 Opening things

- `Ctrl+O` opens files, `Ctrl+Shift+O` a folder. Drag files or folders onto the window to open them.
- `Ctrl+P` (Goto Anything) fuzzy-finds any file in the open folders.
- **File > Open Recent** lists recent files and projects.
- **File > Open Remote** edits a file on another machine over SSH (section 5.2).
- Large files (over `large_file_mb`, 50 MB by default) open in a reduced mode without highlighting, minimap detail or word wrap so they stay responsive. Binary files open in the hex editor (section 2.7).

Try it 2 minutes

Open a folder with `Ctrl+Shift+O`. Press `Ctrl+P` and type three letters of a file name. Press `Enter`. That is most of what you need.

1.4 The command line

```
apex                          open (or focus) the editor
apex file.rs                  open a file in the running editor
apex file.rs:120:8            open at line 120, column 8
apex .                        open a folder as a workspace
apex a.rs b.rs "my notes.md"  several files; spaces need quotes
apex --new-window file.rs     a second, independent window
apex --wait file.rs           return when the tab is closed (EDITOR=apex --wait)
apex --diff old.rs new.rs     compare two files side by side
apex --version | --help
```

A running instance is found through `instance.json` in the per-user runtime directory; a second invocation hands over its paths and exits. Set `APEX_RUNTIME_DIR` to isolate instances. `–log debug` or `APEX_LOG=debug` prints logging to stderr.

1.5 Where things live

Platform	Settings, keymap, themes, snippets, dictionaries
Linux	`~/.config/apex`
macOS	`~/Library/Application Support/cx.dixon.Apex`
Windows	`%APPDATA%\dixon\Apex\config`

Sessions, recovery snapshots and macros go to the data directory (`~/.local/share/apex` on Linux, the same folder as above on macOS, `%LOCALAPPDATA%\dixon\Apex\data` on Windows). **Help > About** shows the exact paths; `APEX_CONFIG_DIR` and `APEX_DATA_DIR` override them.

1.6 Updates

Help > Check for Updates asks GitHub for the latest release. If it is newer, you see its version, its release date and a link to the release notes, and **Download** opens the installer for your platform in your browser. Installed with Homebrew or from repo.dixon.cx? Update through that instead. Apex never checks by itself.

Done when you can open a folder, find a file with `Ctrl+P`, and open a file from the command line.

CHAPTER 2

Editing

TL;DR

Everything you expect from an editor, plus multiple cursors that are real cursors: typing, deleting, pasting and undo act on all of them at once.

Fifteen minutes, then practice

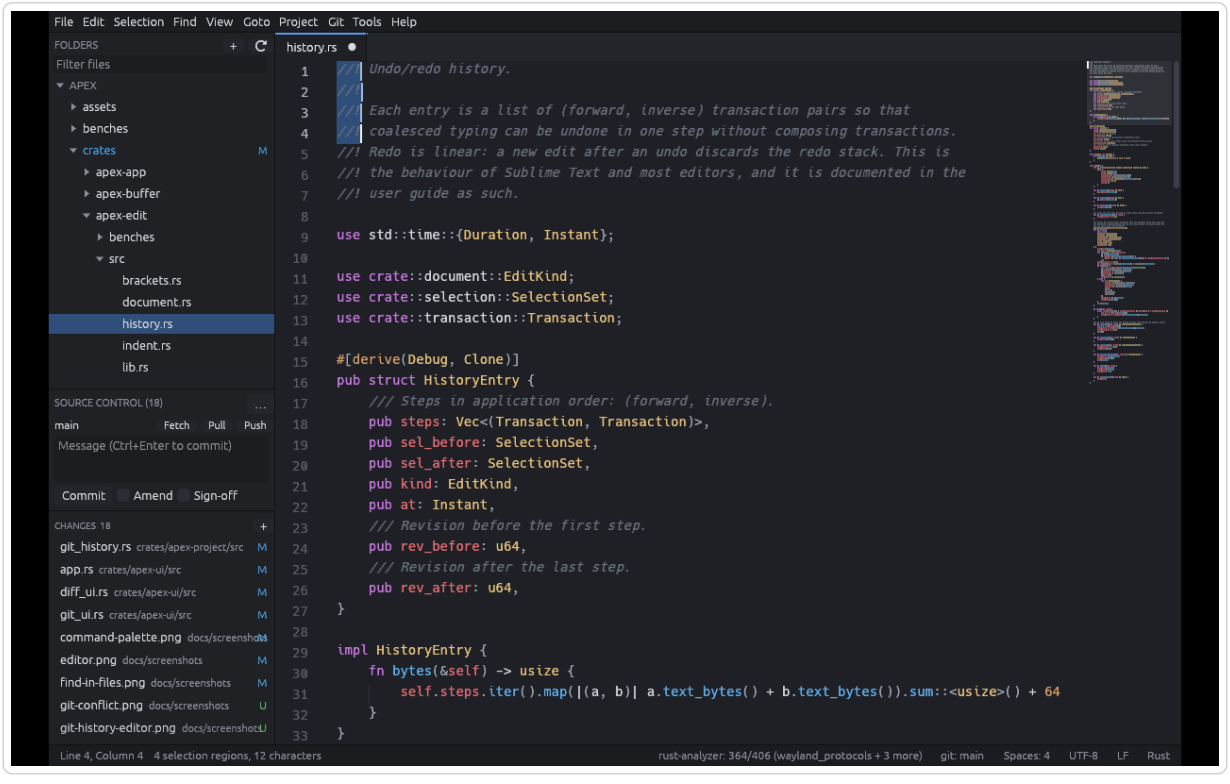
2.1 Everyday keys

Keys	Action
`Ctrl+Z` / `Ctrl+Shift+Z`	Undo / redo. Typing coalesces into one step; selections come back with each step.
`Ctrl+X` `Ctrl+C` `Ctrl+V`	Cut, copy, paste. With nothing selected, the whole line is cut or copied; pasting a copied line inserts it above the caret line.
`Ctrl+Shift+V`	Paste and re-indent to the caret's indentation.
`Ctrl+Alt+V`	Paste from the clipboard history (last 32 copies).
`Ctrl+Shift+D`	Duplicate the line or selection.
`Ctrl+Shift+K`	Delete the line.
`Ctrl+Shift+Up` / `Down`	Move lines up or down.
`Ctrl+J`	Join lines.
`Ctrl+Enter` / `Ctrl+Shift+Enter`	New line below / above, keeping indentation.
`Ctrl+/` / `Ctrl+Shift+/`	Toggle line / block comment for the file's language.
`Ctrl+]` / `Ctrl+[`	Indent / outdent. `Tab` indents a selection.
`Ctrl+Backspace` / `Ctrl+Delete`	Delete a word backwards / forwards.
`Ctrl+T`	Transpose characters.
`F9` / `Ctrl+F9`	Sort lines (case-sensitive / insensitive). The **Edit** menu also has unique, reverse, shuffle, trim trailing whitespace, sequence insertion and case conversions (`Ctrl+K Ctrl+U` upper, `Ctrl+K Ctrl+L` lower, `Ctrl+K Ctrl+T` title, plus snake, camel and kebab).

Keys	Action
`Alt+Q`	Hard-wrap the paragraph at `wrap_column` (80 when unset).
`Insert`	Overwrite mode (block caret).
`Ctrl+Space`	Completion: words from open files, snippets and the language server. `Tab` or `Enter` accepts, `Escape` dismisses.
`Ctrl+Shift+I`	Format the document with the language server.

Brackets and quotes auto-pair (type the closing one to skip over it, or set `auto_pair = false`). `Backspace` in leading whitespace removes one indent level. Indentation is detected per file; the status bar shows it and the **Spaces: 4** entry switches tabs and spaces and converts.

2.2 Multiple cursors



Four selections edited as one. The status bar counts the selection regions and the characters selected.

Keys	Action
`Ctrl+Click`	Add a caret (or remove the one under the pointer).

Keys	Action
`Ctrl+D`	Select the word under the caret, then add the next occurrence each press. `Ctrl+K Ctrl+D` skips one.
`Alt+F3`	Select every occurrence.
`Ctrl+Shift+L`	Split a selection into one caret per line.
`Alt+Shift+Up` / `Down`	Add a caret on the line above / below (the column is kept across short lines).
`Ctrl+Shift+Alt+Up` / `Down`, `Shift`+right-drag, middle-drag	Column selection.
`Ctrl+U`	Undo the last selection change.
`Escape`	Back to a single caret.
`Ctrl+L`	Select the line (repeat to extend).
`Ctrl+Shift+M` / `Ctrl+M`	Select inside brackets / jump to the matching bracket.
`Ctrl+Shift+J`	Select the indentation block.
`Ctrl+Shift+A` / `Ctrl+Shift+Alt+A`	Expand / shrink the selection by syntax scope.

Pasting with as many clipboard lines as there are carets puts one line at each caret. The status bar shows the caret count and selected characters.

Try it 2 minutes

Put the caret on a variable name. Press `Ctrl+D` three times. Type a new name. Press `Escape`. You have just renamed three occurrences in one undo step.

2.3 Bookmarks

`Ctrl+F2` toggles a bookmark on the line; `F2` / `Shift+F2` jump between bookmarks; `Ctrl+Shift+F2` clears them. Bookmarks move with the text as you edit and are saved with the session.

2.4 Snippets

Type a trigger and press `Tab` (`fn` in Rust, `def` in Python), or choose **Edit > Insert Snippet**. `Tab` and `Shift+Tab` move through the placeholders; `Escape` ends the session. Your own snippets go in `<config>/snippets/<language>.toml`:

```
[[snippet]]
trigger = "log"
description = "Log a value"
body = "println!(\"${1:name} = {:?}\", ${1});$0"
```

Variables such as `$FILENAME`, `$SELECTION`, `$CLIPBOARD` and `$DATE`, and choices like `${1|a,b,c|}`, follow the usual snippet syntax.

2.5 Macros

Tools > Macros: `Ctrl+Alt+R` starts and stops recording, `Ctrl+Alt+Shift+R` replays. A macro is a list of built-in commands and typed text. It can be saved under a name and replayed from the command palette. Macros cannot run programs or scripts.

2.6 Spell checking

Tools > Toggle Spell Check (or `spell_check = true`) underlines misspelt words in prose files, and in comments and strings of code. Right-click a word for suggestions, **Add to dictionary** or **Ignore**. Dictionaries are Hunspell `.dic` word lists found in the system locations or `<config>/dictionaries/<lang>.dic`; the status bar says when none is found.

2.7 The hex editor

Binary files open in a hex editor: offsets on the left, the bytes in hex in the middle, the same bytes as text on the right (`.` for anything unprintable). Rows hold 32, 16 or 8 bytes, whichever fits.

- Click a byte in either column to put the caret there; `Tab` switches columns. The status bar shows the offset and the column.
- In the hex column just keep typing digits: two digits replace a byte and the caret moves on by itself. In the text column each character replaces bytes with its own. Typing past the end appends; nothing is inserted or deleted, so offsets never shift.
- Arrows move by byte and by row, `Home` / `End` along the row, `Ctrl+Home` / `Ctrl+End` to either end of the file, `Shift` or a drag selects.
- Copy gives hex (`de ad be ef`) from the hex column, text from the text column. Undo, Redo and Save work as usual, and Save writes the bytes back exactly.
- `Ctrl+F` finds your search as text and, when it is hex digits, as bytes: `de ad be ef` and `deadbeef` find the same four bytes. Replace is not available here.

> ⚠ **Watch out.** Hex edits are not covered by crash recovery: save as you go. To read a binary file as text instead, use **Reload with Encoding**.

✅ **Done when** you can rename three occurrences of a name with `Ctrl+D` and undo it in one step.

CHAPTER 3

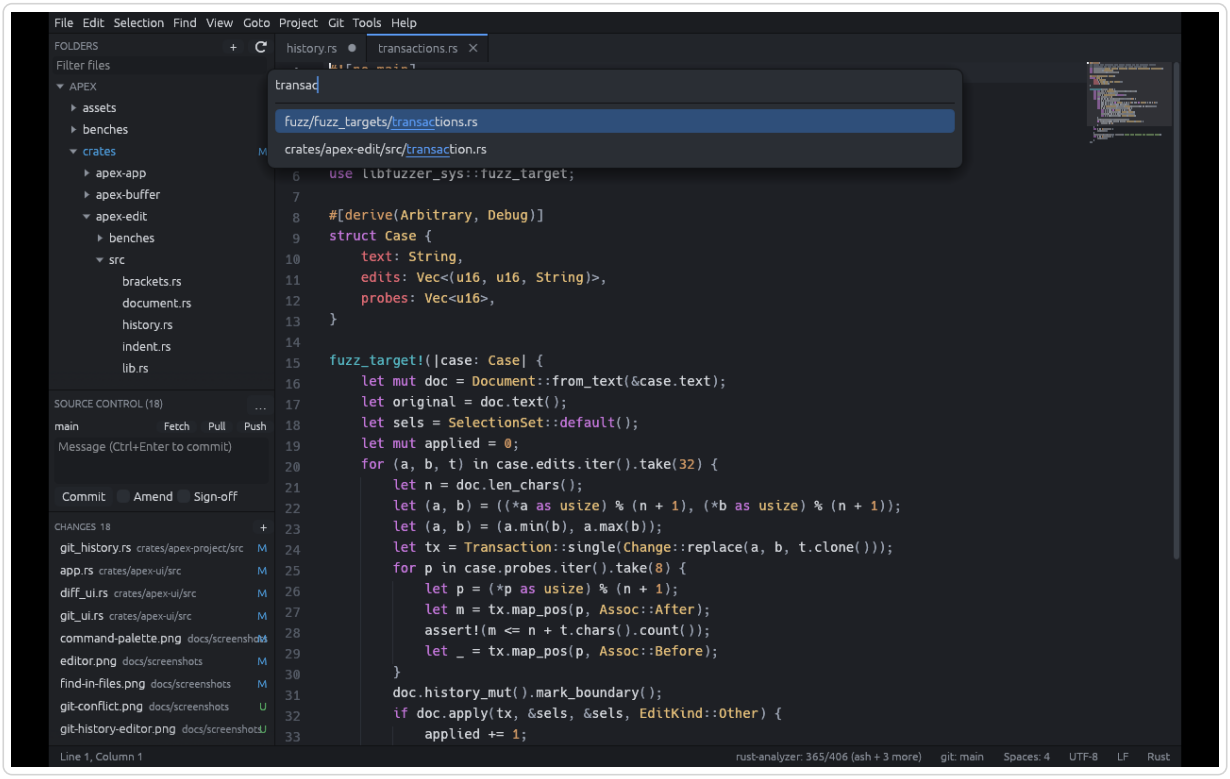
Navigation

TL;DR

One box, `Ctrl+P`, finds files, lines, symbols and commands. Prefixes choose which.

Five minutes

3.1 Goto Anything (`Ctrl+P`)



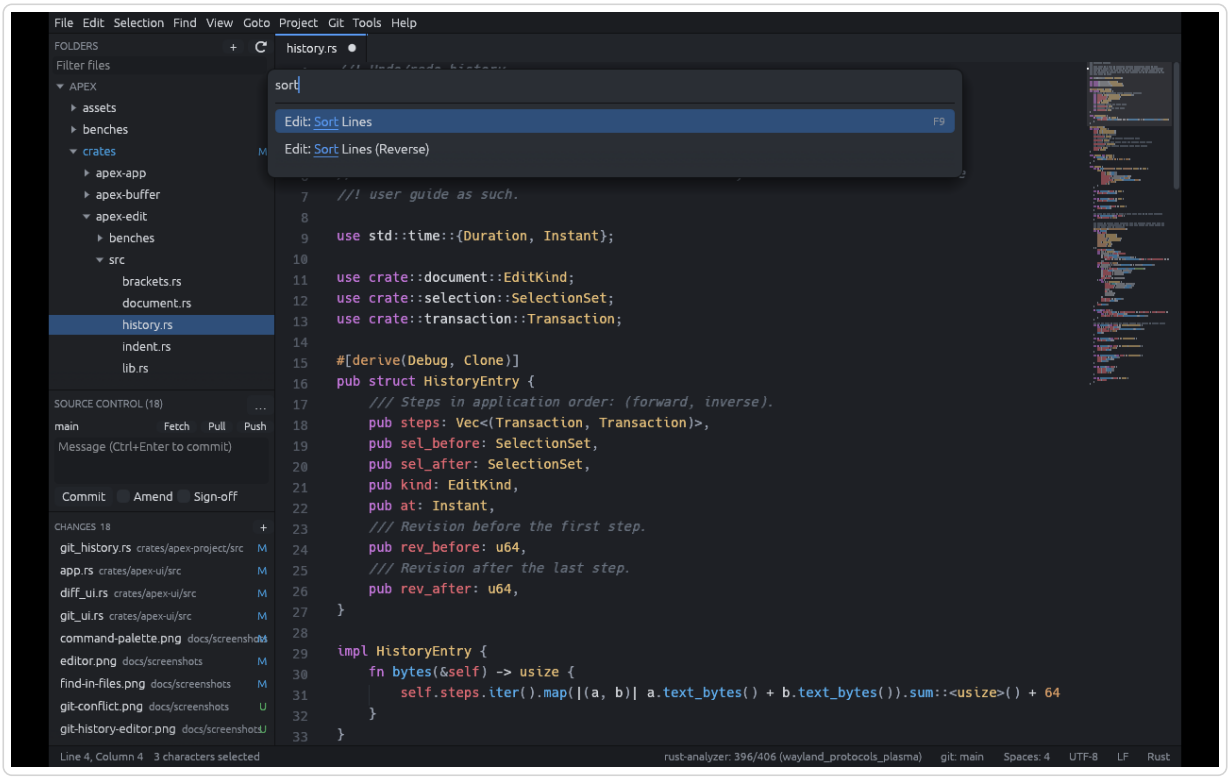
Goto Anything with `transac` typed: fuzzy matches on file names and paths, and the highlighted file previewed behind.

Type part of a file name or path to fuzzy-find it in the open folders. The file is previewed as you move through the list; `Escape` takes you back to where you were.

Prefix	Finds
`:120`	Line 120 (`:120:8` for a column) in the current file
`@name`	Symbols in the current file (`Ctrl+R`)
`#name`	Symbols across the project (`Ctrl+Shift+R`)
`>command`	Commands: the command palette (`Ctrl+Shift+P`)
`file@name`, `file:12`	A symbol or line in another file

`Ctrl+G` is a plain go-to-line dialog.

3.2 Command palette (`Ctrl+Shift+P`)



The command palette filtered to `sort`. Each command shows its shortcut on the right.

Every command with its shortcut, fuzzy-filtered, most recent first. Commands that take an argument prompt for it. Settings, themes, snippets, macros, tasks and git status all have palette entries. Typing a command's full name always picks that command, even when a recently used near-match would otherwise rank higher.

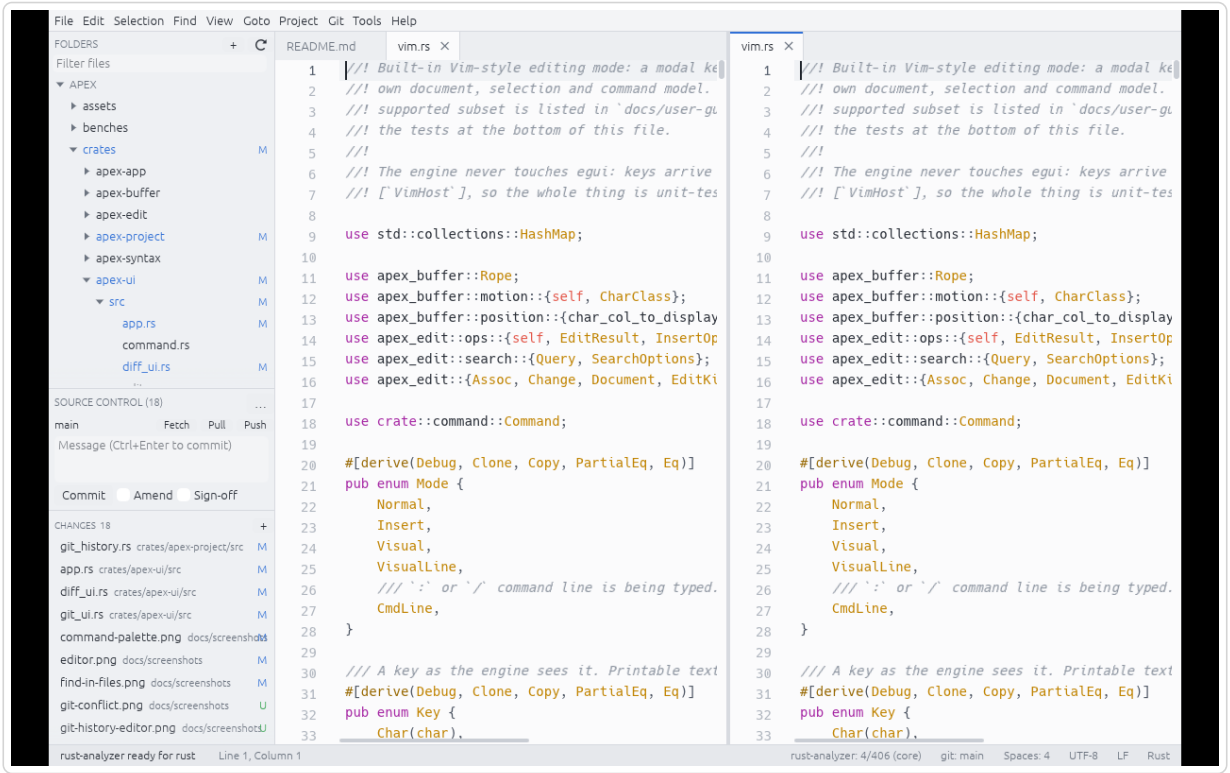
3.3 Moving around

- `F12` go to definition, `Shift+F12` references (language server, or the project symbol index). `Alt+-` / `Alt+Shift+-` go back and forward through the jump history.
- `Ctrl+M` jumps to the matching bracket; `Ctrl+Up` / `Ctrl+Down` scroll without moving the caret; `Ctrl+K Ctrl+C` centres the caret line.

- `Ctrl+PgUp` / `Ctrl+PgDn` and `Ctrl+1` to `Ctrl+9` switch tabs; `Ctrl+Shift+T` reopens the last closed tab.
- Folding: click the gutter arrows or use `Ctrl+Shift+[` / `Ctrl+Shift+]`, `Ctrl+K Ctrl+1` to `3` to fold to a level, `Ctrl+K Ctrl+J` to unfold all.
- The minimap shows the whole file; click or drag it to scroll. Breadcrumbs above the editor show the enclosing symbols.

3.4 Panes and tabs

`Ctrl+\` splits the pane. **View > Layout** has single, two and three columns, two rows and a grid (`Alt+Shift+1`, `2`, `3`, `8`, `5`). `Ctrl+K Ctrl+Left` / `Right` move focus between panes. The same file can be open in several panes with independent scroll positions and selections; edits show up everywhere at once.



The same file in two panes, in the Apex Light theme. Each pane scrolls on its own.

Tabs: drag to reorder, right-click for close others / close to the right, pin, copy path, reveal in the file manager. A file opened from the sidebar with a single click is a *preview* tab (italic title) that the next preview replaces; editing it or double-clicking makes it permanent.

Done when you can reach a file, a line and a symbol from `Ctrl+P` without touching the mouse.

CHAPTER 4

Search and replace

TL;DR

`Ctrl+F` in the file, `Ctrl+Shift+F` across the project. Both support regular expressions, whole word and case options.

Five minutes

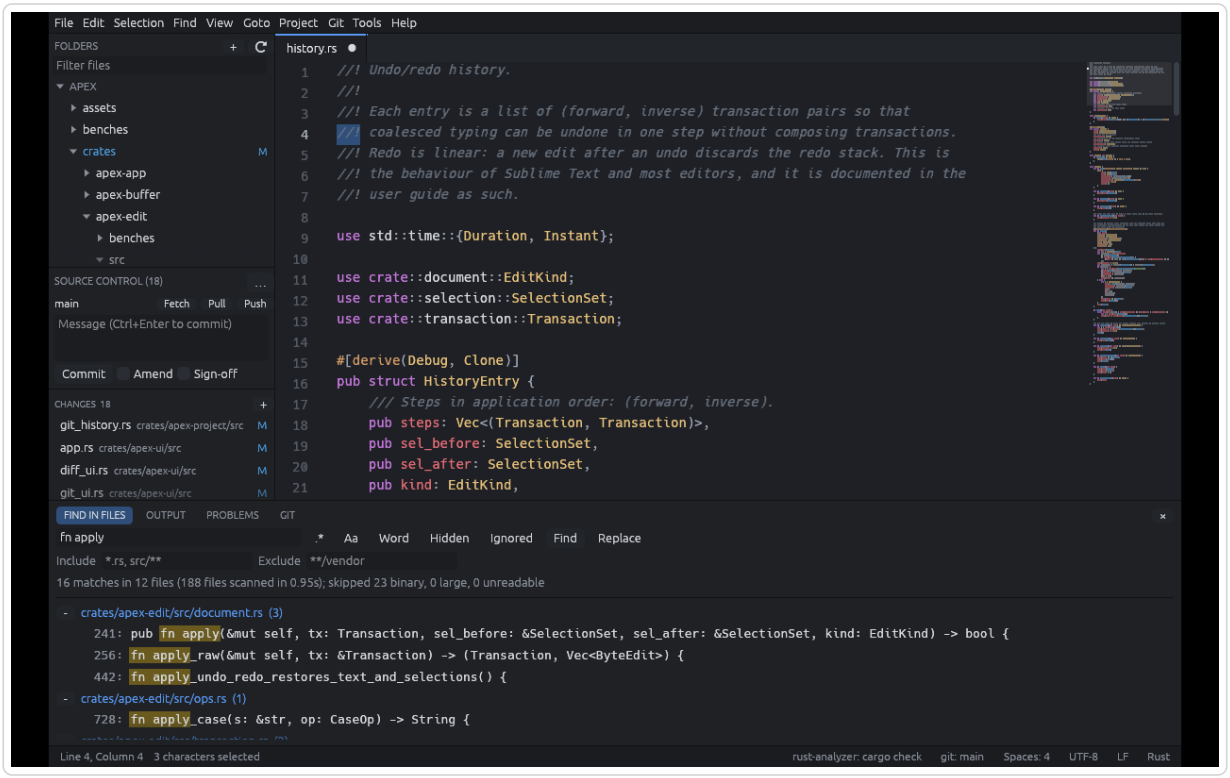
4.1 In the file

The find bar appears above the status bar. Matches are highlighted in the text, the scrollbar and the minimap. `Enter` / `Shift+Enter` and `F3` / `Shift+F3` step through matches; `Escape` returns to the editor at the current match.

Keys	Option
`Alt+R`	Regular expression (Rust `regex` syntax; errors show in the bar)
`Alt+C`	Case sensitive
`Alt+W`	Whole word
`Ctrl+E`	Use the selection as the search text
`Ctrl+F3`	Find the word under the caret
Find in Selection	Restrict the search to the selected range

Replace (`Ctrl+H`) adds a replacement field: `Ctrl+Shift+H` replaces the current match and moves on, `Ctrl+Alt+Enter` replaces all in one undo step. Replacements support `$1` and `${name}` groups in regex mode and preserve case when the search is case-insensitive. **Find All** turns every match into a cursor.

4.2 Across the project



Find in Files: include and exclude globs above, results grouped by file below, every match highlighted.

Find in Files searches the open folders in the background with the same syntax plus include and exclude globs (`*.rs, !target/**`). Results are grouped by file with context lines; click one to open it at the match. Binary files, `exclude_patterns` and `.gitignore` entries are skipped.

Replace in Files previews the count per file and asks first. Every changed file is written atomically; files open in the editor are updated in place and keep their undo history.

Done when you can find a word across the project and replace it in one file with a single undo step.

CHAPTER 5

Projects, the sidebar and sessions

TL;DR

Open folders, not projects. A project file is optional and only adds settings and tasks. Everything you had open comes back next time.

Ten minutes

5.1 Folders

File > Open Folder (`Ctrl+Shift+O`), dragging a folder onto the window, `apex .` or the **+** button in the sidebar add a folder to the workspace. Several can be open at once. The sidebar reads only the directories you expand, so a 100,000-file tree opens instantly; the index behind Goto Anything and Find in Files is built in the background and kept current by a file watcher.

Right-click an entry for new file, new folder, rename, duplicate, delete (to the system bin, with confirmation), copy path, reveal, find in folder. Files ignored by `.gitignore` and `exclude_patterns` are dimmed or hidden. Git status is shown next to changed files.

5.2 Remote files over SSH

File > Open Remote opens a browser for a server. Type a host (`user@server`, or an alias from `~/.ssh/config`), press **Connect**, and click a file. Click a folder to go into it, **Up** to go back. Apex runs your own `ssh`, so your keys, agent, `ProxyJump` and the rest of your SSH config just work. The server needs nothing but `sh`, `cat`, `ls`, `cksum` and `mktemp`.

Open Folder, bottom right of the browser, puts the current folder in a **REMOTE** section at the top of the sidebar instead. Click a folder to expand it and a file to open it. Right-click the top row to **Refresh** it or **Close Remote Folder**. It is a file tree only: no Find in Files, git, rename or delete, and it is not restored after a restart.

- The tab is called `name (host)`.
- **Save** writes back to the server. The file keeps its owner and permissions, and a dropped connection never leaves it half-written.
- **Save As**, and **File > Save to Remote** for any tab, pick a place in the same browser. It asks before replacing a file.
- **Revert** takes the server's copy.
- Every 5 seconds Apex checks each remote file. No unsaved edits: it reloads quietly. Unsaved edits: a note in the status bar; **Save** overwrites theirs, **Revert** takes theirs.

When the connection drops

Nothing pops up. The tab title says `offline`. A save that fails keeps your changes in the editor (and in crash recovery) and finishes by itself when the server answers again, unless the file changed there in the meantime; then you choose. Remote tabs do not come back after a restart.

Passwords and passphrases

Keys and the agent are tried first. If a password or passphrase is needed, a dialog asks, over the browser. Cancel or a wrong answer shows "Permission denied"; press **Connect** to try again.

> **Your data.** Apex remembers the password so it does not ask again and so background checks keep working. It is held **only in memory**, in pages locked so they are never swapped to disk (and kept out of crash dumps on Linux). It is never written to disk, the session, recovery or logs.
> It is kept **while a file or folder from that server is open or the browser shows that server**. When the last one closes, the memory is overwritten with zeros, released, and the shared SSH connection is closed. Quitting wipes it too. A password the server rejects is forgotten at once. One-time codes are never kept.
> Background checks and automatic retries never ask; without a remembered password they just show `offline`.
> The text box you type into belongs to the GUI toolkit. Apex clears it when the dialog closes but cannot promise no stray copy was left while you typed. If that matters to you, use a key with an agent.

> **Watch out.** The first connection to a new server records its host key without asking. If a server's key later changes, Apex refuses to connect; check with the server's owner, then fix `~/.ssh/known_hosts`.

5.3 Project files

Project > Save Project As opens a native Save panel and suggests `<name>.apex-project`. Setting `native_file_dialogs = false` uses a path prompt instead. The project file contains:

```
# Apex project file. Safe to commit: it holds no session state.
folders = ["."]

[settings]
tab_width = 2

[[tasks]]
name = "Check"
command = ["cargo", "check"]
problem_matcher = "rustc"
```

Opening the project file restores the folders and applies the `[settings]` overrides on top

of your own. Tasks from a project file run only after you trust the workspace (chapter 8).

5.4 Sessions and recovery

When you quit, the open folders, tabs, selections, scroll positions, pane layout, bookmarks and window size are saved and restored on the next start. Unsaved buffers are kept too.

Every few seconds unsaved changes are snapshotted to the data directory. After a crash or power cut the next start offers to restore them.

> **Your data.** Files changed on disk by another program are noticed (on focus and every two seconds). An unmodified buffer reloads silently; a modified one asks whether to reload or keep your version. Saving over a file that changed since it was loaded asks first. Apex never silently overwrites someone else's edit.

Done when you know where your folders, sessions and recovery snapshots come from, and what Apex keeps when you quit.

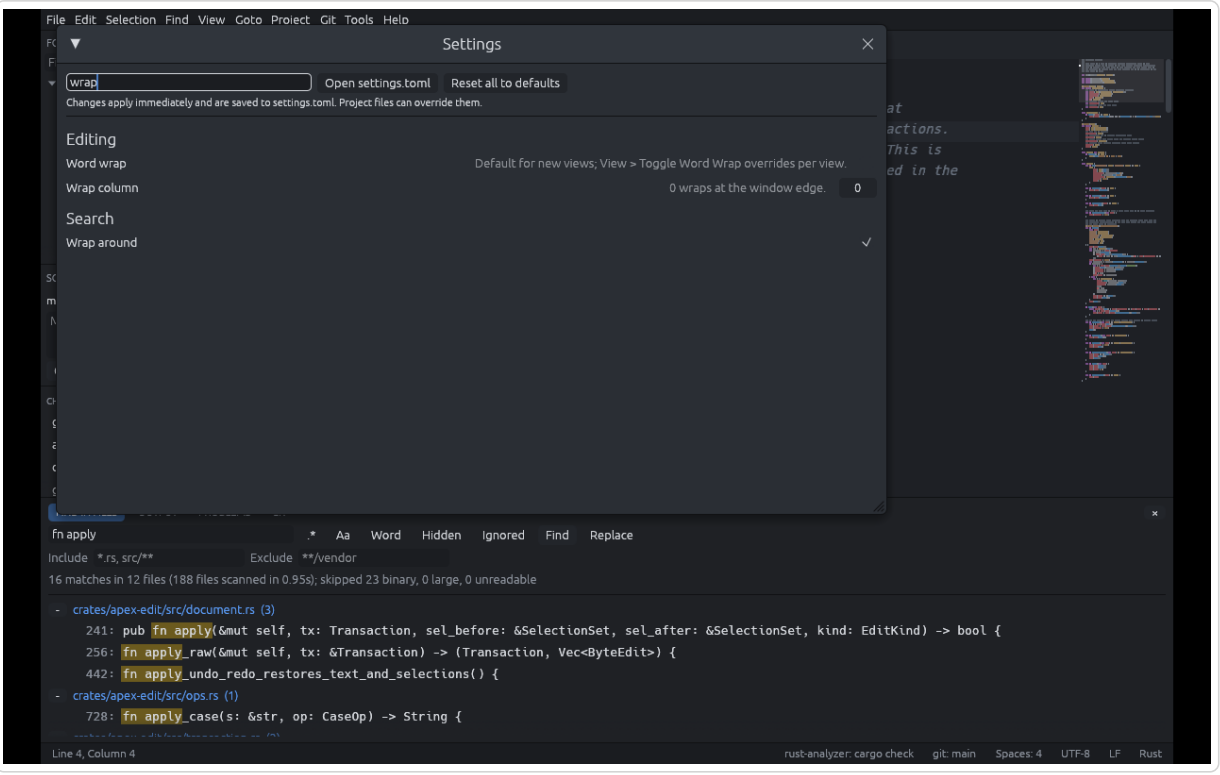
CHAPTER 6

Configuration

TL;DR

`Ctrl+,` opens a searchable settings panel. Everything is plain TOML in the configuration directory. There is no scripting; the editor only ever reads these files.

Ten minutes



The settings panel filtered to `wrap`. Changes apply at once and are saved to `settings.toml`.

6.1 Settings

Changes apply immediately and are written to `settings.toml`, which stores only the keys you changed. **Open settings.toml** edits the file directly; saving it reloads the settings, and problems (unknown keys, wrong types, out-of-range values) are reported in a dialog rather than ignored.

Layers, lowest first: built-in defaults, `settings.toml`, the `[settings]` table of the open project file.

Open, folder selection and Save As use native system dialogs by default, including Open Project and Save Project As. Set `native_file_dialogs = false` to use the in-app path prompt instead.

```
theme = "Apex Light"
font_family = "JetBrains Mono"
font_size = 15
tab_width = 2
word_wrap = true
rulers = [80, 120]
trim_trailing_whitespace_on_save = true
vim_mode = true
spell_check = true
spell_language = "en_GB"
native_file_dialogs = true      # false: an in-app path prompt with Tab completion
exclude_patterns = ["target", "node_modules", "*.min.js"]

[lsp_servers]
rust = ["rust-analyzer"]
python = ["pyright-langserver", "--stdio"]
```

6.2 Key bindings

Tools > Open Keymap File creates `<config>/keymap.toml`, merged on top of the defaults:

```
[[bindings]]
keys = "primary+k primary+t"      # chords are space-separated
command = "edit.title_case"
context = "editor"                # global, editor, find, overlay, sidebar

[[bindings]]
keys = "f5"
command = "build.run"
arg = "Cargo: test"               # optional command argument

[[macos]]                         # platform sections: [[linux]], [[windows]], [[
    ↪ macos]]
keys = "primary+shift+f"
command = "find.in_files"
```

`primary` is `Ctrl` on Linux and Windows and `Cmd` on macOS. **Tools > Keyboard Shortcuts** lists the effective bindings and reports conflicts; the status bar shows a pending chord.

6.3 Themes

Eight themes are built in. **View > Theme** switches, and `theme_mode = "auto"` follows the system's dark or light setting. User themes are TOML files in `<config>/themes/`:

```
name = "My Theme"
dark = true

[ui]
background = "#1e2127"
text = "#d7dae0"
accent = "#61afef"
```

```
[editor]
background = "#1e2127"
caret = "#528bff"
selection = "#3e4451"

[syntax]
keyword = { fg = "#c678dd", bold = false }
string = { fg = "#98c379" }
comment = { fg = "#5c6370", italic = true }
```

Unknown keys are reported; missing keys fall back to the base theme.

6.4 Dictionaries

Hunspell `.dic` word lists are looked up in:

- `/usr/share/hunspell` and `/usr/share/myspell`
- `~/Library/Spelling` (macOS)
- `<config>/dictionaries/`
- `/usr/share/dict/words`

Words you add go to `<config>/dictionaries/user.txt`.

Done when you have changed one setting in the panel and found it in `settings.toml`.

CHAPTER 7

Git

TL;DR

Apex reads your repository on its own and runs *your* installed `git` for anything that changes it. The Source Control panel in the sidebar does staging and commits; the Git menu has everything else: diffs, blame, branches, log, stash, conflicts, push and pull, GitHub pull requests and commit-history editing.

Twenty minutes

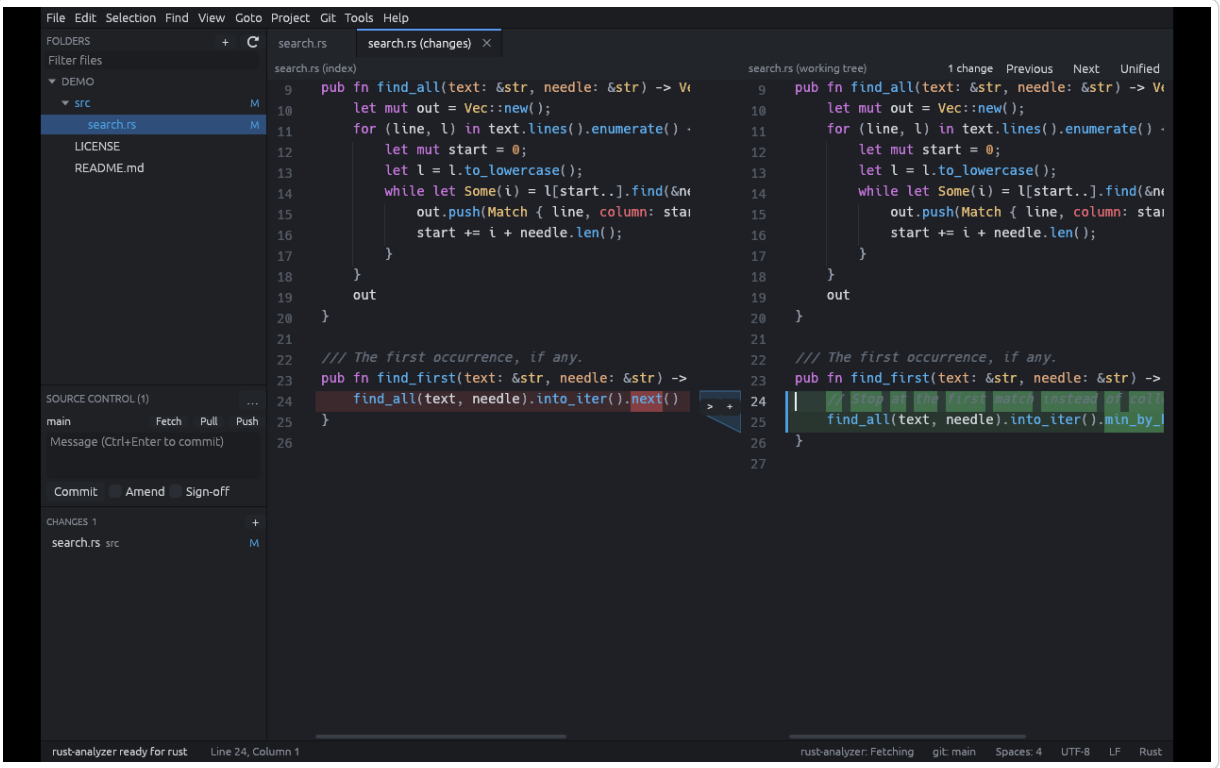
Your data. Apex stores no git credentials or tokens. Push and pull use your credential helper and SSH agent. When git or ssh asks for a password or passphrase, Apex shows a dialog, hands the answer to git once, and forgets it. GitHub features use the GitHub CLI (`gh`), which holds your login. Commands that run `git` ask you to **trust the folder** first, because git runs the repository's hooks.

7.1 What you see without doing anything

- **Gutter marks** beside the line numbers show added, modified and deleted lines against the index. They update as you type and the minimap shows them too. `git_gutter = false` turns them off.
- **Status bar**: the branch (or a short commit id when detached), commits to push and pull, the branch's pull request, and any merge or rebase in progress.
- **Folder tree**: changed, added, untracked and conflicted files are marked.

Goto > Next Change / **Previous Change** jump between hunks. **Revert Change** restores the hunk under the caret after confirming, as an undoable edit. **Compare with Saved / Index / HEAD** opens a side-by-side diff. **File History** lists the commits that touched the file; choosing one shows it.

7.2 The Source Control panel



A side-by-side diff: the index on the left, your working file on the right, and the strip between them linking each change.

The lower half of the sidebar shows the repository of the active file: the branch and how far it is ahead of or behind its upstream, the commit box, and changed files grouped as **Merge conflicts**, **Staged changes** and **Changes**. Drag the divider to resize it; `git_sidebar = false` or **Git > Toggle Source Control Panel** hides it.

- Click a file to preview it, double-click to open its changes. Hover a file for **+** stage, **–** unstage and discard; right-click for the full list.
- **Commit** (or `Ctrl+Enter` in the message box) commits the staged changes. With nothing staged it offers to commit all tracked changes. `Up` and `Down` in an empty box recall earlier messages; the number beside Sign-off is the summary line's length (50 recommended, 72 at most).
- **Amend** rewrites the last commit; **Sign-off** adds a `Signed-off-by` trailer.
- **Undo Last Commit** removes the last commit after confirming, keeps its changes staged and puts its message back in the box.
- Your hooks run as usual; their output is in the GIT tab.

Try it 2 minutes

Edit a file in a repository. Click **+** next to it in the Source Control panel. Type a message, press `Ctrl+Enter`. Open **Git > Show Log** and see your commit at the top.

7.3 Side-by-side diffs

Open Changes (double-click a file in the panel), **Open Staged Changes**, the Compare commands and `apex --diff old new` show the two versions next to each other.

- Both sides have syntax colours. Removed lines are tinted on the left, added lines on the right, changed words more strongly. A strip between the sides links each change.
- When the right side is your file, it is the real document: edit, undo and save there as usual.
- The strip's buttons act on one change: **>** replaces it with the left side (undoable), **+** stages it, **–** unstages it.
- Long unchanged stretches are folded on both sides, keeping three lines of context; **Expand / Collapse** in the header toggles this.
- **Next / Previous** jump between changes; **Unified** shows the same comparison as a unified diff. Closing the tab closes both sides.

7.4 Staging

- **Stage File / Unstage File** for the active file; **Stage All Changes / Unstage All Changes** for the repository.
- **Stage Change at Caret** stages just the hunk under the caret; **Stage Selected Lines** stages only the changed lines your selections touch (several cursors work). Save first: staging uses the file on disk.
- **Discard File Changes** restores the file from the index after confirming; for an untracked file it offers to delete it.

7.5 Fetch, pull and push

Apex only talks to the network when you ask, or when `git_auto_fetch_minutes` is above 0.

- **Fetch** fetches every remote and prunes deleted branches.
- **Pull** follows `git_pull_mode`: `ff-only` (default) refuses to create a merge commit, `rebase` replays your commits, `merge` merges.
- **Push** pushes to the upstream branch. A branch without one is published to `origin` after asking.
- **Sync** pulls then pushes. **Force Push (with Lease)** replaces the upstream branch after confirming and refuses if the remote changed since your last fetch.
- Progress appears in the status bar; **Cancel Git Command** stops a slow transfer.

7.6 GitHub and other code hosts

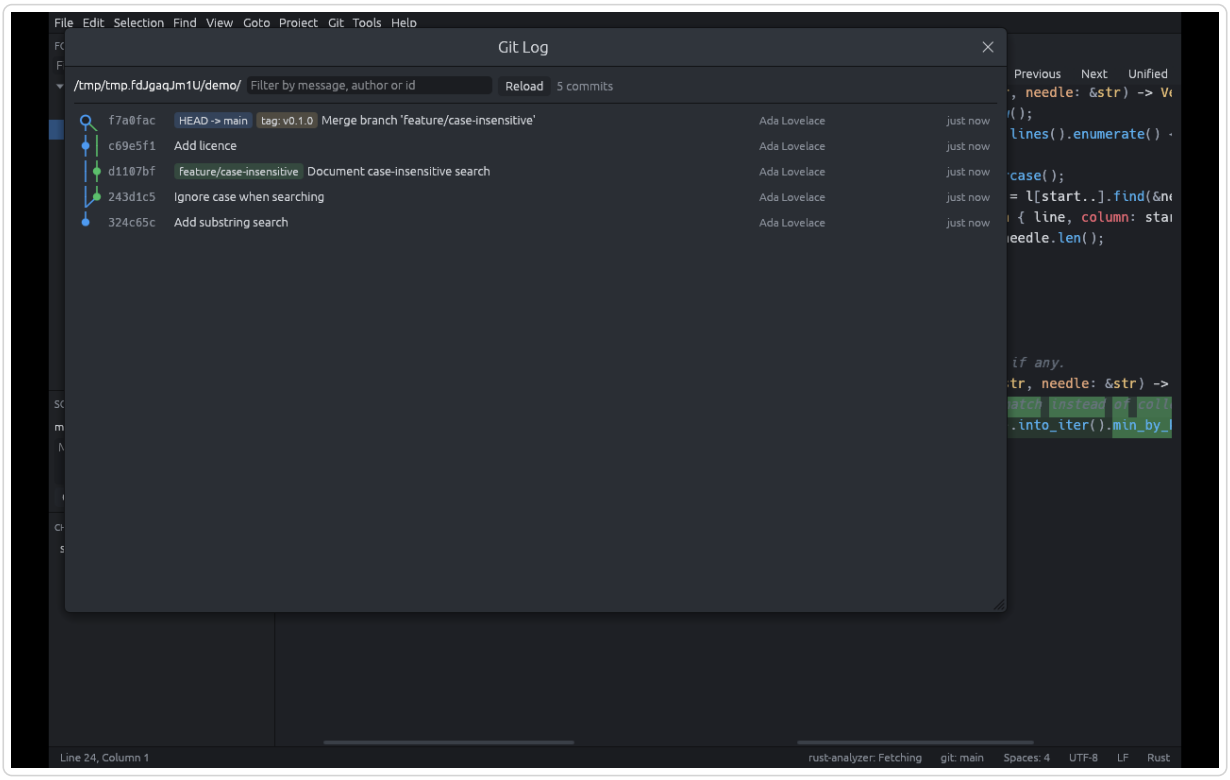
Git > Remote builds links from the repository's remote (GitHub, GitLab, Bitbucket, Codeberg and Gitea) and opens them in your browser: **Open File on Remote**, **Copy Permalink**

(with the selected lines), **Open Commit / Branch on Remote**, **Compare Branch on Remote**. Apex makes no network request for these.

Git > GitHub uses the GitHub CLI. Install it and run `gh auth login` once.

- **Create Pull Request**: title and description filled in from the branch's commits, base branch, draft. Unpushed commits are pushed first.
- The status bar shows the branch's pull request, its checks and review state. It refreshes after you push or create a pull request, and on **Refresh Pull Request Status**; Apex does not poll GitHub.
- **Pull Requests** lists open ones to check out, show the diff or open in the browser. **Pull Request Checks** lists the checks.

7.7 Branches, log and stash

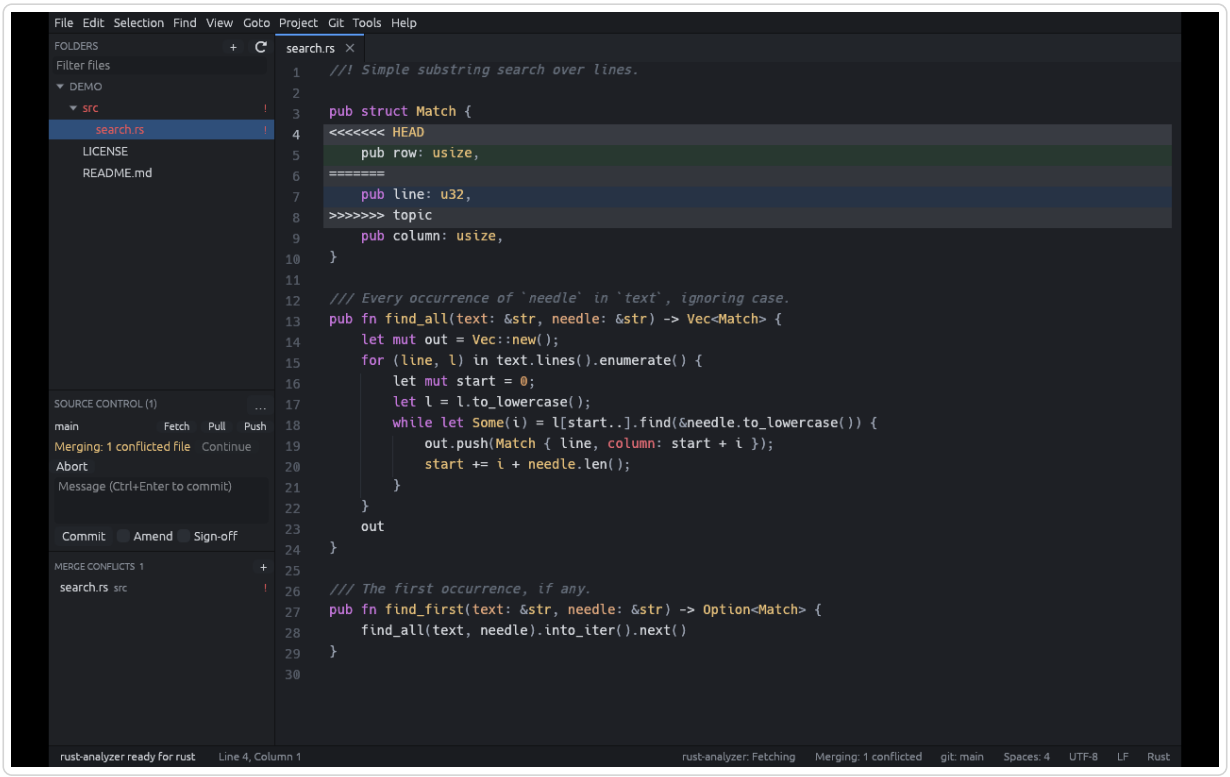


Show Log: every branch and tag with a lane graph. Type to filter by message, author or id.

- **Switch Branch** (or click the branch name in the panel) lists local branches and remote ones without a local copy; the first entry creates a new branch. Git refuses to switch when your changes would be overwritten: stash them first.
- **Create / Rename / Delete Branch** and **Merge Branch into Current** are under **Git > Branches**. Deleting refuses a branch with unmerged commits.
- **Show Log** opens every commit of every branch and tag with a lane graph. Type to filter; `Enter` shows a commit; right-click to copy its id, create a branch there, check it out, cherry-pick it, revert it, or open it on the remote.
- **Line History of Selection** shows how the selected lines changed over time.

- **Stash Changes** stashes everything including untracked files; **Stashes** lists them with Apply, Pop, Show and Drop.

7.8 Merge conflicts



A conflict block, tinted in the editor. The panel lists the file under Merge conflicts, with Continue and Abort.

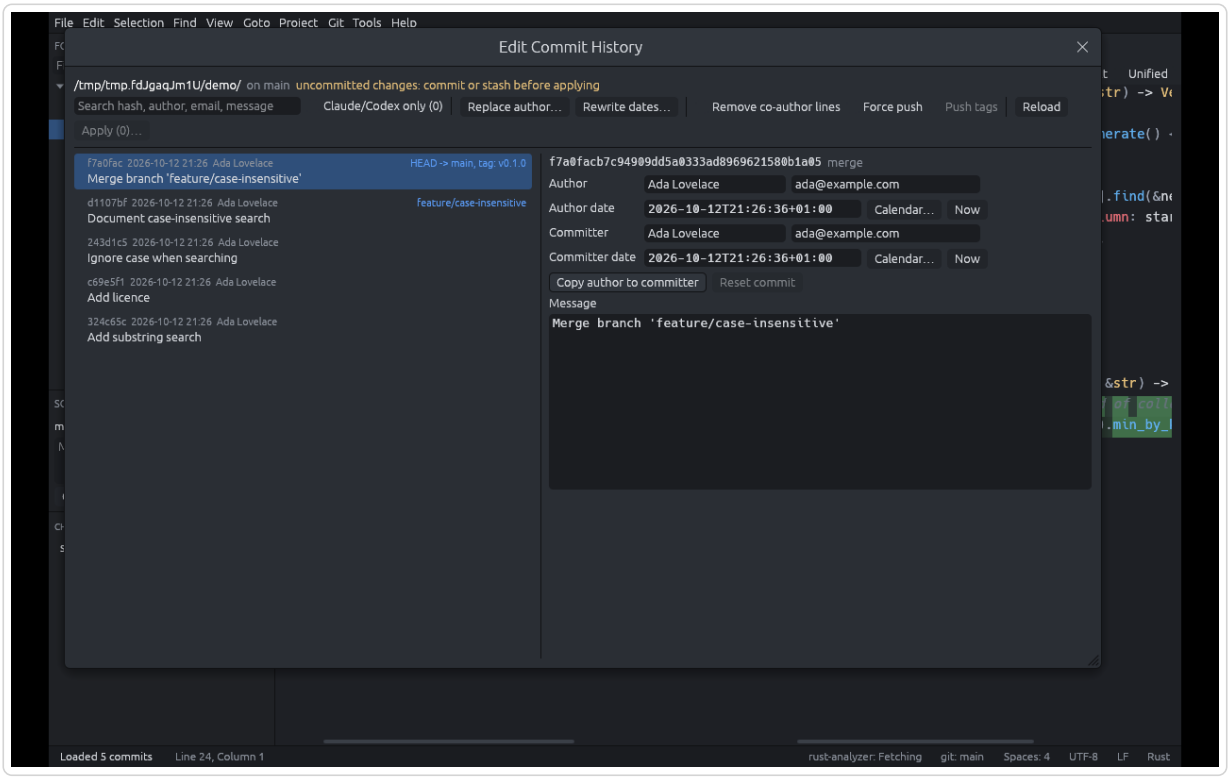
When a merge, rebase, cherry-pick or revert stops on conflicts, the status bar and the panel say so, and the conflicted files are listed under **Merge conflicts**.

1. Conflict blocks are tinted: ours, theirs and the marker lines differ. **Next / Previous Conflict** move between them.
2. With the caret in a block, **Accept Current Change**, **Accept Incoming Change** or **Accept Both Changes** (**Git > Conflicts** or right-click) replace the block. Each is an ordinary undoable edit.
3. Save, then **Mark Resolved** to stage the file (it warns if markers are left).
4. When nothing is conflicted, **Continue** in the panel finishes the merge or moves the rebase on. **Abort** returns to where you started, after confirming.

7.9 Blame

Git > Toggle Blame (`Ctrl+K Ctrl+G`) adds a gutter column with the author and age of the commit that last changed each line, shaded so newer changes stand out. Unsaved edits show as *Not committed*. Hover for the full message; click to open the commit in a read-only tab. `git_blame_inline = true` also writes it after the caret line.

7.10 Edit Commit History



Edit Commit History: the commits on the left, the selected commit's author, committer and message on the right. Nothing changes until you apply.

Git > Edit Commit History opens an editor for the metadata of every commit in the repository, newest first. Nothing changes until you apply.

- Select a commit to edit its author and committer (name, email, date) and its message. **Calendar** picks a day and time; **Now** sets the current time. **Reset commit** undoes the edits to one commit.
- Search filters by hash, author, email or message. **Claude/Codex only** lists commits that mention Claude, Codex or Anthropic.
- **Replace author** sets a new author on every matching commit. **Rewrite dates** gives every commit a random time inside a daily window across a range of days, keeping their order. **Remove co-author lines** drops `Co-authored-by:` trailers and AI attribution blocks.
- **Apply** lists the commits that will change and asks first. The repository must be clean. Apex tags the current HEAD as `apex-backup-<time>`, rewrites local branches and tags with `git filter-branch`, then cleans up. To go back: `git reset –hard apex-backup-<time>`.
- **Force push** then pushes every branch with `–force-with-lease`; you must type `FORCE` to confirm.

⚠ **Watch out.** Rewriting history gives every changed commit, and every commit after it, a new hash. Anyone who has the old history must re-clone or reset. Try it on a

throwaway clone first, and coordinate with anyone else using the repository.

7.11 The GIT output tab

Every command Apex runs, with its output, is shown in the **GIT** tab of the bottom panel (**Git > Show Git Output**). Commands run one at a time in the background; **Git > Cancel Git Command** stops the running one. Git never prompts on a terminal.

Done when you have staged a change, committed it from the Source Control panel and found it in **Git > Show Log**.

CHAPTER 8

Build tasks

TL;DR

`Ctrl+B` runs the build for the folder; `Ctrl+Shift+B` picks a task. Errors land in the Problems tab and are underlined in the editor.

Five minutes

Output streams into the bottom panel; `Escape` or **Cancel Build** stops the process and its children.

8.1 Where tasks come from

- **Detected**: `Cargo.toml` gives cargo build, test, run, clippy and check; `go.mod`, `package.json`, `Makefile`, `CMakeLists.txt` and Python project files give their usual tasks. Single files get "run file" tasks for Python, Rust, shell, JavaScript, Go and Lua.
- **Project file** `[[tasks]]` entries:

```
[[tasks]]
name = "Test (verbose)"
command = ["cargo", "test", "--", "--nocapture"]
cwd = "crates/apex-edit"          # relative to the workspace root
env = { RUST_BACKTRACE = "1" }
shell = false                     # true runs command[0] through sh -c / cmd /C
timeout_secs = 600                # 0 = no limit
problem_matcher = "rustc"         # rustc, gcc, go, python, generic, or a regex
```

A custom matcher is a regular expression with named groups `file`, `line`, optional `col` and `message`.

8.2 Trust

Your data. Tasks from a project file can run any program, so they are disabled until you trust the workspace: the first attempt shows the command and asks. **Project > Trust Workspace** manages the decision, which is remembered per folder in your data directory and never inside the repository. The first line of the Output panel is always the exact command and directory used. Commands never go through a shell unless `shell = true`, and output is stripped of terminal escape sequences before it is shown. A task inherits Apex's environment, as a command typed in a terminal would.

8.3 Problems

Matched errors and warnings appear in the **Problems** tab with file, line and message; click one to jump there. Language-server diagnostics share the same list. The status bar shows the count.

Done when `Ctrl+B` runs a task and its errors appear in the Problems tab.

CHAPTER 9

Language servers

TL;DR

Optional. If a known server is on your PATH and the folder is trusted, it starts when you open a file of its language and gives you completion, diagnostics, hover, go to definition, rename and formatting. Everything else works without one.

Five minutes

Your data. Servers act on the whole folder, and some run its code: rust-analyzer runs a Rust project's build scripts. So a server starts only in a folder you have trusted (**Project > Trust Workspace**). In an untrusted folder the status bar says the server was not started; trust the folder and it starts at once.

9.1 Starting a server

With `lsp_enabled = true` (the default) a known server starts the first time you open a file of its language, if it is on your PATH:

Language	Server
Rust	`rust-analyzer`
Python	`pyright-langserver`, `pylsp`
TypeScript, JavaScript	`typescript-language-server`
Go	`gopls`
C, C++	`clangd`
Lua, Bash	`lua-language-server`, `bash-language-server`
JSON, CSS, HTML	the `vscode-*-language-server` packages
YAML, Markdown, TOML	`yaml-language-server`, `marksman`, `taplo`
Java, Kotlin, Swift	`jdtls`, `kotlin-language-server`, `sourcekit-lsp`
PHP, Ruby, C#	`intelephense`, `solargraph`, `omnisharp`

Other servers, or different commands, go in settings:

```
[lsp_servers]
zig = ["zls"]
python = ["ruff", "server"]
```

The status bar shows the server's state and progress. **Tools > Restart Language Server** restarts it; a server that exits three times in a row is left stopped and its last stderr lines are shown.

9.2 What you get

- Diagnostics underlined and listed in Problems; hover shows the message.
- Completion merged with word and snippet completions.
- Hover documentation and signature help.
- `F12` definition, `Shift+F12` references, document and workspace symbols in Goto Anything.
- **Rename Symbol** (`F7`), **Code Actions** (`Ctrl+K Ctrl+I`), **Format Document** (`Ctrl+Shift+I`, also `format_on_save`). Edits to files not open in the editor are previewed and confirmed first.

Servers run as child processes with only the workspace folders as roots. The client makes no network connections.

✅ **Done when** the status bar shows your language server running in a trusted folder, or you know why it is not.

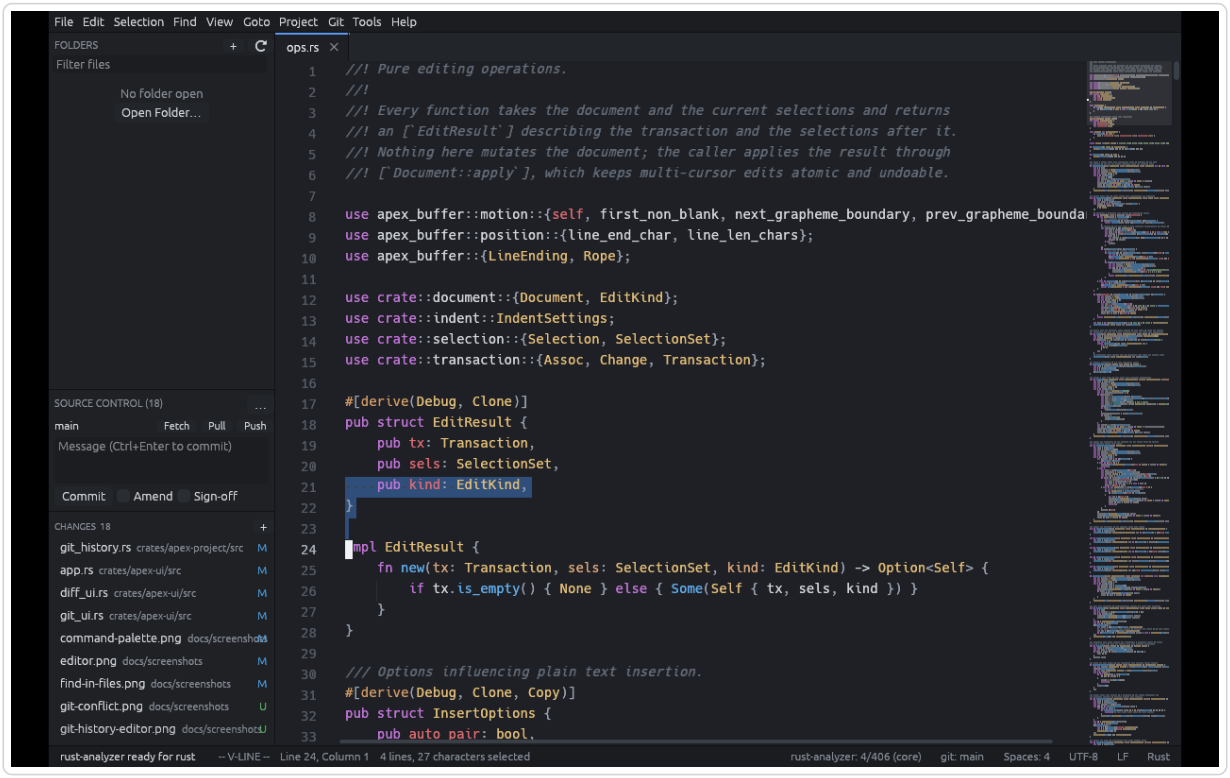
CHAPTER 10

Vim mode

TL;DR

A modal layer over Apex's own editing model: normal, insert, visual and visual-line modes, operators, text objects, registers, the dot command and `:` commands. Not a full Vim; what is listed here is what works.

Five minutes, if you already know Vim



Vim mode in visual-line mode. The status bar shows the mode.

Turn it on with `vim_mode = true`, or for the session with **Tools > Toggle Vim Mode**. The status bar shows the mode, partially typed keys, and the command line. The mouse, menus, the palette, Find and all `Ctrl` shortcuts keep working.

10.1 Modes and motions

`i` `a` `I` `A` `o` `O` enter insert mode; `Esc` returns (the insert session is one undo step). `v` and `V` are visual and visual-line; `o` swaps the ends. `:` opens the command line; `/` and `?` search.

Motions: `h j k l`, arrows, `0 ^ $ |`, `w W b B e E ge`, `gg G`, `{ }`, `%`, `f F t T ; ,`, `H M L`, `Ctrl+F` `Ctrl+B` `Ctrl+D` `Ctrl+U`, `n N * #`, marks with `m` and `'`. Counts work with every motion.

10.2 Operators and text objects

`d c y > < = gu gU g~`, each with a motion or text object, doubled for whole lines (`dd`, `3dd`), and on the selection in visual mode. Shortcuts: `x X s S D C Y J r ~ p P`; `R` is replace mode.

Text objects: `iw aw iW aW ip ap is as`, brackets `( { [ <`, quotes `" ' `` ` ``, and tags `it at`.

10.3 Registers, repeat, command line

`"a` to `"z` named (capitals append), `"0` last yank, `"1` to `"9` deletes, `"_` black hole, `"+` and `"*` the system clipboard. `.` repeats the last change; `u` undoes, `Ctrl+R` redoes.

`:w`, `:q`, `:wq`, `:x`, `:e file`, `:N`, `:s/pat/rep/g`, `:%s/.../.../g`, `:noh`, `:set wrap|nu|rnu|spell`, `:sp`, `:vs`, `:bn`, `:bp`, `:reg`. `:q` closes the tab; `:qa` quits.

> ⚠ **Watch out.** No visual block mode (use column selection), no `q` macros (use the editor's recorder), no `:g`, mappings or `.vimrc`.

✅ **Done when** you can switch Vim mode on and off and know which Vim features are missing.

CHAPTER 11

Troubleshooting

TL;DR

Most problems are one of these. Each entry is a symptom followed by the thing to try.

As long as it takes

Where are the logs?
: Apex logs to stderr. Run `apex --log debug` from a terminal. There is no telemetry and no log file unless you redirect one.

The window is blank or does not open.
: Try OpenGL with `WGPU_BACKEND=gl`, or `WGPU_BACKEND=vulkan` with a software Vulkan driver on machines without a GPU. On Wayland without decorations, `WINIT_UNIX_BACKEND=x11` forces X11.

A settings dialog on start-up.
: It lists keys in `settings.toml` or the project file that were unknown or out of range; those were ignored and everything else applied.

"No dictionary found".
: Install a Hunspell dictionary (`hunspell-en-gb` on Debian and Ubuntu) or put a word list at `<config>/dictionaries/<lang>.dic`.

A language server does not start.
: Check its program is on the PATH of the shell that launched Apex; desktop launchers often have a shorter PATH. Configure the full path under `[lsp_servers]`.

A build task is greyed out.
: Tasks from a project file run only after **Project > Trust Workspace**.

The Open or Save As dialog does not appear (macOS).
: If the system reports a panel failure, Apex opens a path prompt instead. Enter the file or folder path; `Tab` completes it. Cancelling keeps your unsaved work. Set `native_file_dialogs = false` to use the path prompt immediately for every request. A stalled macOS panel service can still block the system dialog. If file dialogs also hang in other applications, preserve unsaved work and restart macOS.

Fonts show boxes.
: Set `font_family` to an installed font with the glyphs you need.

Keys do nothing in the editor.
: Vim mode may be on (the status bar says `-- NORMAL --`): press `i` to insert or turn it off under **Tools**. A pending chord such as `Ctrl+K` is shown in the status bar; `Escape` cancels it.

Recovering work after a crash.
Start Apex; it offers the snapshots it kept. They live in the data directory under `recovery/`.

Resetting everything.
Delete the configuration and data directories listed in **Help > About**.

Reporting a problem.
Include the version (`apex --version`), the platform, the `--log debug` output around the problem and, for crashes, the `RUST_BACKTRACE=1` output. Nothing is sent automatically.

✅ **Done when** the thing works again. If it does not, you have the version, platform and `--log debug` output for a report.

Open Apex. Find the thing. Change it.
Save. Close Apex.

uncomplex. Complicated things. Thoughtfully simplified.