A MINDMAP STUDIO GUIDE

Thinking in Maps

A practical guide to mind mapping — the technique and the canvas — with MindMap Studio.

Dann Bleeker Pedersen

Generated 2026-07-08
mindmap-studio.struktureretsundfornuft.dk

Contents

Foreword

Why a book about drawing bubbles and lines?

A mind map looks like the simplest thing in the world: a word in the middle, some branches, a few more words. People sketch them on napkins. So why a whole book?

Because the simplicity is the point, and the discipline behind it is easy to miss. A mind map is a thinking tool disguised as a doodle. It externalises the shape of an idea -- what depends on what, what sits beside what, what is still missing -- so your working memory stops juggling and your eyes can do the work instead. Done well, it turns a vague "I should plan this" into a structure you can argue with.

This book is in two voices at once. The first is about **the technique** : how to grow a map that actually helps you think, when to branch, when to stop, how to keep a big map from collapsing into noise. The second is about **the tool** : MindMap Studio , a small, fast, local-first app that runs in your browser, keeps your maps on your own machine, and asks for no account and no network. Every technique in the book is shown end-to-end in the app, with the exact keys and controls you press.

Who this is for

You don't need any prior experience with mind mapping, and you don't need to be a designer or a "visual thinker" -- a phrase that does more harm than good. If you can write a bulleted list, you can build a mind map, and this book will take you from a single node to a map you'd happily present.

If you're coming from MindManager, XMind, or a wall of sticky notes, you'll find the moves familiar and the friction lower. MindMap Studio is deliberately small: it does brainstorming and structuring well, and leaves project-management gravity to other tools.

How to read it

- **Part 1 -- Getting started** gets a real map on your screen and explains the few ideas the whole app rests on.
- **Part 2 -- Building maps** is the craft: structure, enrichment, and keeping large maps navigable.
- **Part 3 -- Sharing your thinking** covers getting the map *out* -- exporting, presenting, and running a session with other people.
- **Part 4 -- The practice** returns to pure method: the six jobs you'll most often bring to a map -- brainstorming, studying, meetings, decisions, plans, root-cause -- and the shape of a good session for each.
- The **appendices** are reference: a keyboard card, the file-format details, and where to read more.

Read Part 1 in order. After that, jump to whatever you need. The map you build in Chapter 1 carries through the book, so it's worth following along in the app as you go.

Everything autosaves. You can't lose your place. Let's begin.

Your first map

From a blank canvas to a real structure in five minutes

Open MindMap Studio . The first time, you land on the **Start screen** -- a capture box up top, templates and worked examples below it. You could type a topic into the capture box and be on a fresh map in one keystroke, but for this chapter take the deliberate route: pick **Blank** .

You now have a single node that says *Untitled map* . That node is the **root** -- the one idea everything else hangs off. Double-click it (or just start typing) and give it a real subject. A map about planning a conference might have a root that simply says *Conference* .

The three keys that do everything

Almost all of map-building is three keys. Select the root, then:

- **Enter** adds a *sibling* -- another node at the same level. From the root, the first Enter gives you your first branch.
- **Tab** adds a *child* -- a node one level deeper, attached to the node you have selected. This is how a branch grows outward.
- **Delete** removes the selected node (and everything under it).

That's the whole loop: Tab to go deeper, Enter to stay at the same level, type to label, Delete to prune. Try it. Select *Conference* , press **Tab** , type `Venue` . Press **Enter** , type `Programme` . Press **Enter** , type `Budget` . You have three branches. Select `Venue` , press **Tab** , and add `Shortlist` , `Site visit` , `Contract` as its children. Within a minute you have a small tree.

> A fourth key is worth learning early: **Shift+Tab** outdents -- it promotes the selected node one level back toward the root, the undo for an over-eager Tab. (And **Ctrl+Enter** adds a child of the selected node without leaving the keyboard's home row -- the same move as Tab, from inside an edit.)

The keys are the fast path, but the canvas reaches back. Hover a node (or select it) and a small **+** appears on its right edge -- click it to add a child -- and a second **+** below it to add a sibling. Either way you land straight in the new node, ready to type, so a hand on the mouse never has to find the keyboard. The root only offers the child **+** : nothing sits beside the centre of a map.

There is also no separate "rename" step. To relabel any node, **double-click** it, press **F2** with it selected, or -- fastest -- just **start typing** : the first letter you press replaces the old label and the rest follows. The same three gestures work whether the node is brand new or one you made an hour ago.

Capturing in a hurry

The three keys are for *building* a map; sometimes you just need to *dump* something in before it escapes. Two shortcuts help. The **Quick add** box in the toolbar takes a topic and an **Enter** -- type, return, type, return -- and each line lands as a node without your touching the canvas: the fastest

way to empty a head full of ideas into a map and sort them out later. And you can **drop** straight onto the canvas -- drag a link from your browser, or selected text from anywhere, onto the map and it becomes a **floating topic** where you let go (a link arrives already titled after its address, no network needed). Capture first, structure second; the dragging-into-shape of Chapter 3 is a calmer job once the ideas are already on the page.

A third capture works from *inside* the editor: type ***/*** at the start of a topic and a small command menu opens -- add a child, a task, a date, a note, a marker -- so a hand that's typing never has to leave for the toolbar. (Appendix A collects these editor accelerators.)

Both of those capture *into the map you're on* . For the thought that belongs to no map at all -- or to one you don't want to open right now -- there's the **Inbox** (in the Panels menu, as **Inbox (quick capture)**): jot a line into its box, press Enter, and it's held -- across maps, across reloads -- until you deal with it. Each held item has a small **-> map** button that files it onto whichever map you have open (it lands as a floating topic, ready to place) and a **×** to discard it. It's the app's back-of-an-envelope: capture now, decide where it lives later. Chapter 8 builds a weekly ritual on exactly this.

Moving around

Click any node to select it; the arrow keys walk you between parent, child and siblings. Drag the canvas background to pan; scroll or pinch to zoom. If you ever lose the map off the edge of the screen, the **Fit** button in the toolbar frames the whole thing again.

One more canvas gesture earns its keep: **double-click an empty patch** of background and a fresh topic appears there, already in edit mode. It lands as a *floating topic* -- unattached to the tree -- which is exactly what you want when an idea arrives before you know where it belongs. Drop it down, name it, and connect it later; Chapter 3 covers the dragging-into-shape.

You can't lose it

MindMap Studio saves continuously to your browser's local storage (IndexedDB) as you type. Close the tab, reopen it tomorrow, and the map you were working on is exactly where you left it. You never *have* to press Save -- it has already happened. (If you prefer a map to live as a file on disk -- in a synced folder, say -- you can link it to a `.mmst` file and the same autosave writes through to it; the user guide covers working with files.) There is also no server: the map never leaves your machine unless you explicitly export it, which is Chapter 6.

Undo is your safety net

Made a mess? **Ctrl+Z** undoes; **Ctrl+Y** (or **Ctrl+Shift+Z**) redoes. Undo is model-aware -- it reverses the actual change to the map, not just a visual tweak -- so you can experiment with structure freely and walk it back if a branch turns out wrong.

Home base: the Start screen

Once you've built a map or two, the **Start** button (top-left) takes you **home** -- a Start screen that's part launcher, part library. A **capture box** at the top turns a stray thought into a new map in one line; below it your saved maps sit as cards, alongside the **templates** and worked **examples** to start from, an **import** drop zone, and links to learn the app (this book included). It's the calm front door to the workbench -- somewhere to land, pick up an existing map, or start a fresh one, rather than always opening straight onto the canvas. Press **Start** any time to return; choose a map (or start one) and you're back on the canvas, where the rest of this book happens.

The very first time you open a fresh map, a small **"3 things to try"** card sits in the corner: double-click a topic to rename it, press **Tab** to add a child, press **Ctrl/Cmd + K** for anything. It is a one-time nudge -- close it with the **x** , or just make your first edit and it retires itself for good. You will not see it nagging on every map.

Now you try

Start a **Blank** map and build something real to you in five minutes -- a trip, a project, a decision you're weighing. Name the root, then grow three or four branches with a couple of children each, using only **Tab** (deeper), **Enter** (same level), and typing. Delete a branch and **Ctrl+Z** it back, just to feel the safety net. Then close the tab and reopen it: the map is exactly where you left it. That loop -- Tab, Enter, type, undo, and it already saved itself -- is the whole foundation the rest of this book builds on.

Three captures worth trying once: from the **Start** screen, type a title into the capture box and press Enter -- you're on a fresh map in a single line; while building, throw three half-formed ideas through the **Quick add** box (type, Enter, type, Enter) to feel how *capturing* and *structuring* are two different gears; and drop one stray thought into the **Inbox** , close the map, open a different one, and file it there with its **-> map** button -- proof that a thought no longer needs a destination before you're allowed to write it down.

Try the mouse-only path too, so you know it is there when your hand is already on the trackpad. Hover a branch, click its **+** to add a child, and type the label that appears under your cursor. Then **double-click an empty patch** of canvas: a floating topic drops where you clicked, ready to name. Rename any node three ways to feel them -- **double-click** it, press **F2** , and on a third just **start typing** -- and notice you never had to reach for a menu.

That's enough to build maps. The next chapter slows down for a moment to explain *what* you've been building -- the handful of ideas the rest of the app rests on.

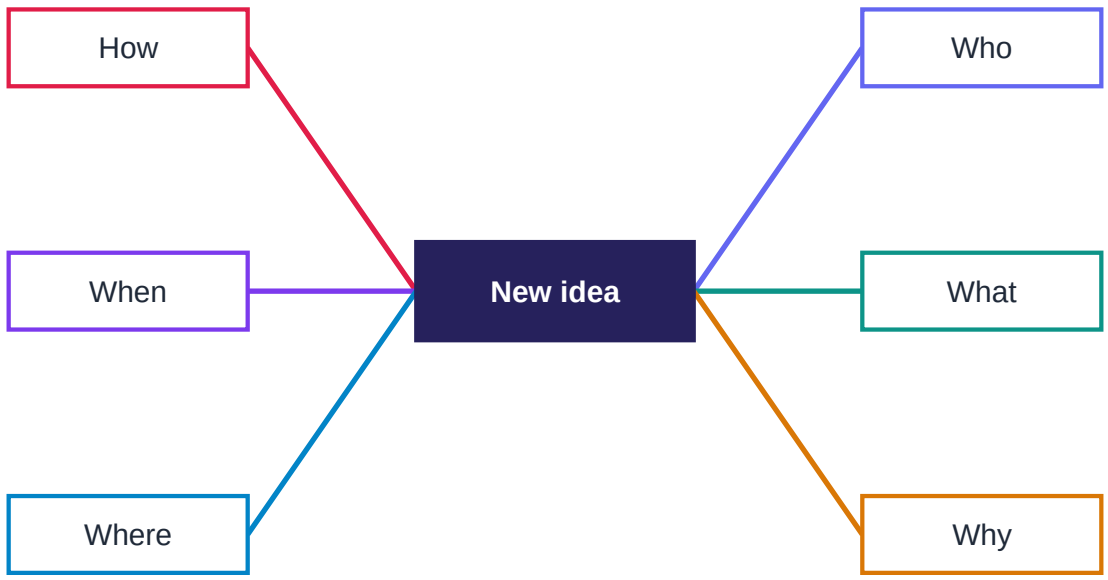
The anatomy of a map

The parts of a map, and why thinking in them works

You can use MindMap Studio without ever reading this chapter. But twenty minutes here will make every later feature feel obvious instead of arbitrary, because they all act on the same small set of parts -- and it will make your maps *better* , because the parts come with a method. This chapter covers both: what a map is made of, and the handful of disciplines that separate a map that helps you think from a decorated list.

The shape: one centre, many branches

A mind map is a tree with a single root in the middle and branches radiating outward. The classic starting point is a central idea surrounded by the six questions every plan eventually has to answer:



A mind map: one central idea, six branches — the Brainstorm template.

This is exactly what the built-in **Brainstorm** template gives you (you'll meet the other templates in Chapter 7). Each branch can grow its own children, and theirs can grow more, as deep as the thought goes. The radial shape isn't decoration: putting the subject in the centre keeps every branch equidistant from it, so no single line of thinking dominates just because it happens to be at the top of a list.

Why a map beats a list

It's worth being precise about what the shape buys you, because the answer explains most of the method that follows.

A map takes the load off your working memory. You can hold perhaps four to seven things in

your head at once; a plan of any size has more parts than that. Written as a list, the parts you aren't currently reading are gone -- you re-derive them every time you scroll. Laid out as a map, *everything is visible at once, in its place* . Your eyes do the remembering, and the headspace that was spent juggling is freed up for the actual thinking: noticing what's missing, what clashes, what connects.

A map keeps relationships visible. A list has exactly one relationship: *comes after* . A map has several at a glance -- *belongs under* , *sits beside* , *is far from* , *points at* . When you notice that a risk on one branch is really the same issue as a constraint on another, that noticing *is* the work, and it happens because the two were on the same page in the first place. Linear notes hide those collisions; radial ones invite them.

A map defers order until you've earned it. The cruellest thing about a blank document is that it demands sequence immediately -- something has to be the first line. But at the start of a piece of thinking you don't *know* what comes first; sequencing is a conclusion, not a starting point. A map lets you put an idea down where it roughly belongs and decide its rank later, which is why a map is the right first surface and the finished document is better written *from* the map than instead of it.

A word with a place is easier to recall than a word in a row. You remember where things are -- it's why you can find the mustard in your own fridge in the dark. A topic that lives at the end of a particular branch, in a particular colour, in a particular corner, gets that spatial memory for free. The same fact as the fourth bullet of the second section has almost nothing to hang on to.

None of this is magic, and none of it requires believing any grand theory of the brain. It's ergonomics: the map is a surface shaped like the problem -- part-whole structure with exceptions -- instead of a surface shaped like a page.

The node: the only real object

Everything on the canvas is a **node** . A node has:

- a **topic** -- the text you see;
- an optional set of **markers** -- small icons or tags (Chapter 4);
- an optional **note** -- longer prose attached behind the node (Chapter 4);
- an optional **image** ;
- and **children** -- the nodes hanging off it.

The root is just a node that happens to have no parent. A leaf is just a node with no children. There is no special "task" type or "milestone" type -- MindMap Studio keeps the model deliberately small. That smallness is why import, export and undo all behave predictably: there are fewer kinds of thing to get wrong.

The things that aren't nodes

A small family of features draws on the canvas without being part of the tree:

- A **boundary** is a soft, shaded box drawn around a node and everything beneath it, grouping a branch visually ("everything in here is Phase 1").

- A **relationship** is an arrow drawn from one node to another, expressing a link that the tree structure can't -- a dependency, an influence, a "see also" across branches.
- A **summary** bracket, a **callout** bubble, and the free **background shapes** are the rest of the family -- annotations and frames you'll meet in Chapters 3 and 4.

The first two carry most of the weight and are covered in Chapter 3. The point for now: the tree carries the *hierarchy* , and everything else carries the *exceptions* and *commentary* around it.

The canonical model

Under the surface, your map is a plain data structure -- a root node, its children, each node's topic and notes and markers. MindMap Studio treats that structure as the single source of truth. The colourful canvas is one *view* of it; the outline panel (Chapter 5) is another; an exported Markdown file is a third. Edit the map on the canvas and the underlying model updates; every other view follows.

This is worth internalising because it explains the app's most reassuring property: **what you see is always backed by real data you can take with you.** A map is never trapped in a proprietary blob. Chapter 6 shows you how to round-trip it through Markdown, JSON, OPML and more -- losslessly, in the case of JSON.

The method in five rules

Everything this book teaches about *technique* -- as opposed to the tool -- compresses into five rules. They date back to the hand-drawn tradition of mind mapping, they survive contact with real work, and every one of them exists to protect the advantages of the shape described above.

1. Keywords, not sentences. A topic should be one to three words -- `Venue shortlist` , not `We need to shortlist some venues by Friday` . This isn't a style preference. A short label can be *read at a glance while looking at something else* , which is what keeps the whole map scannable; a sentence has to be read on purpose, and twenty of them turn your map back into the document you were escaping. Short labels also stay *open* -- `Budget` invites more thoughts underneath it; a full sentence sounds finished and quietly closes the branch. Put the sentence in a **note** behind the node (Chapter 4) if it matters; the canvas gets the keyword.

2. One idea per node. If a label contains an "and", it's usually two nodes -- `Catering and AV` will want to grow in two directions, and can't. Splitting feels pedantic for a second and pays off for the life of the map: single ideas can be moved, marked, filtered, and connected independently. Compound ones can't.

3. Let the branches ask questions. The strongest first-level branches are the questions the subject has to answer -- *who, what, when, where, why, how* for a plan; *strengths, weaknesses, opportunities, threats* for a position; *start, stop, continue* for a retrospective. A branch that names a question pulls answers out of you; a branch that names a category just sits there. When a map stalls, check whether its branches are still asking anything.

4. Structure carries the logic; arrows carry the exceptions. Put "is part of" in the tree, by

placement. Save relationship arrows (Chapter 3) for the handful of links the hierarchy genuinely can't express -- the dependency, the tension, the "these two collide". A map where placement means nothing and arrows do all the work has thrown away its best instrument; a map with no arrows at all has usually just not been looked at hard enough.

5. Prune. A finished map is the *smallest* one that still carries the argument. Merge duplicates, delete the branch that turned out to be someone else's problem, collapse the detail that only mattered while you were finding it. The discipline is the same as editing prose: what you remove sharpens what remains. (Nothing is lost -- Chapter 6's version history keeps what you cut.)

Hold these loosely -- a map for your own eyes can break every one of them and still work. But when a map goes to other people, or has to survive more than a week, the rules are what keep it legible.

Two gears: diverge, then converge

The rules above are about the *state* of a map. The method also has a *rhythm* , and it's the same rhythm every creative discipline discovers: **generating and judging are different modes, and doing both at once does both badly.**

In the **divergent** gear, the only goal is more: capture every idea as a node, put it anywhere, don't evaluate, don't tidy, don't even parent things correctly -- floating topics and misplaced branches are fine. Speed matters because judgement kills the association chains that produce the non-obvious ideas; the moment you stop to fix a label, the next thought escapes.

In the **convergent** gear, you switch loyalties from *more* to *true* : drag things under the branch where they belong, merge the duplicates, prune the noise, name the clusters that emerged, and only now decide what matters. Restructuring isn't overhead -- it's the half of the thinking where conclusions form (Chapter 3 gives you the moves).

The practical habit: **name the gear you're in.** Alone, that can be as simple as a timer -- ten minutes of pure capture, then shift. In a group it has to be said out loud, because one person judging while others generate silences the room (Chapter 7 builds a whole workshop rhythm on this). Most disappointing maps are the product of trying to do both gears at once; most good ones are two clean passes.

When not to map

A method you can't say "no" for isn't a method. Mind maps are the right surface for thinking that is *part-whole shaped with exceptions* -- plans, analyses, decisions, subjects being learned. They are the wrong surface for at least three jobs:

- **True sequences.** A checklist you execute top to bottom, a recipe, a runbook -- if the order *is* the content, a list is simply better. (If a map develops a long chain of single children, it's telling you it wants to be a list; the timeline layout in Chapter 5 is the halfway house.)
- **Heavy comparison.** Choosing between four options across six weighted criteria is a *table* -- rows, columns, and sums. Map the option space to discover the criteria, then move the scoring to a spreadsheet.

- **Finished prose.** A map is scaffolding. When the thinking is done, write the document *from* the map (Chapter 6's exports make that direct) -- don't ship the scaffolding and call it the building.

Knowing when to reach for a different tool is part of what makes the map trustworthy when you do reach for it.

Now you try

Open a worked example (**+ New... -> Examples** -- the *SWOT (worked)* or *Product launch plan* both qualify) and name the parts out loud: which node is the **root** , which are **branches** , where's the **boundary** , where's the **relationship** arrow. Then prove the "one model, many views" idea to yourself -- open the **Outline** panel (Chapter 5) and watch the same tree appear as an indented list, or export to **Markdown** (Chapter 6) and read your map as plain text. Same data, three faces. Now **rename a node on the canvas** and look again: the outline and the Markdown already show your edit, because the canvas writes straight to the one model every view reads from -- nothing to "save", nothing to sync.

Then run the method's most valuable exercise once, deliberately. Take the map you built in Chapter 1 and audit it against **rule 1** : find every label longer than three words and compress it to a keyword, moving anything that mattered into a note. `Book the venue before end of March` becomes `Venue booking` with the deadline in the note (or, later, as a real due date -- Chapter 4). Now stand back and *feel* the difference: the map reads at a glance where it used to need reading. Do the same audit for **rule 2** -- split any node hiding an "and" -- and watch the split halves immediately want different children. Ten minutes of this, once, and the keyword habit sticks for good.

Finally, feel the **two gears** . Set a five-minute timer and add nodes about something you're actually planning -- any branch, any order, no fixing, no judging. When it rings, switch: spend five minutes only dragging, merging and pruning, adding nothing new. Notice how different the two passes feel -- and how much better the map is for having kept them apart.

With the parts named and the rules in hand, the rest of the book is learning what you can do with them -- and Chapter 8 returns to pure method, applying these rules to the six jobs you'll most often bring to a map.

Structuring ideas

Re-parenting, grouping, diagramming, and the links a tree can't hold

A first draft of a map is rarely in the right shape. A branch you started under *Venue* turns out to belong under *Budget* ; two ideas you thought were separate turn out to be the same. Restructuring is not a failure -- it *is* the thinking. This chapter is about moving things around without fear.

Drag to re-parent

To move a node -- and everything beneath it -- to a new home, just **drag it onto its new parent** . Drop `Catering` from under *Venue* onto *Budget* and the whole sub-branch travels with it, re-coloured to match its new branch. There is no cut-and-paste dance; the drag *is* the move. Because the underlying model updates atomically, **Ctrl+Z** puts it back if you misjudged the drop.

Two modifier keys change what a drag means. **Shift-drag onto empty canvas** *detaches* the branch -- it becomes a floating topic instead of snapping back home. **Ctrl-drag onto another topic** drops a *copy* of the branch there, leaving the original where it was -- the quick way to fork a checklist you're about to adapt.

A good habit: build fast and loose first -- get every idea onto the canvas as a node, wherever -- then spend a second pass dragging things into the structure that emerges. Trying to get the hierarchy right while you're still generating ideas slows both down.

Copy a branch -- here, or in another map

Dragging *moves* a branch; sometimes you want a *copy* -- the same sub-tree in two places, or a structure you built in one map reused in another. **Right-click** a topic and choose **Copy branch** ; then right-click where it should go and choose **Paste branch here** to graft a fresh copy -- the node and everything beneath it -- under that parent. Every pasted node gets a new identity, so the copy is genuinely independent: edit it and the original doesn't budge.

The quietly useful part is that the clipboard **crosses maps** . Copy a branch in your *Q3 Plan* , open your *Q4 Plan* , and paste it in -- a standard checklist, a risk list, a team structure you keep re-using no longer has to be rebuilt each time. It's the middle ground between the drag (which only moves things within one map) and a full file import (which brings in a whole map): a way to carry just the piece you want from one map to the next.

Putting siblings in order

Within a branch, order is information: a reader assumes the top item matters most, so make the order say what you mean. **Ctrl+Shift+Up / Down** moves the selected node among its siblings when one deliberate nudge is all it takes. And when the right order is *mechanical* -- alphabetical, by priority, by date -- don't nudge it by hand: the command palette (**Ctrl+K** , Chapter 5) offers **Sort children A -> Z** , **by priority** , **by due date** , and **by progress** , each a one-shot reorder of the selected topic's direct children in a single undo step. It's a sort, not a standing rule -- the map keeps

the order you sorted into until you change it, so sort by due date for the review and drag things back afterwards if the argument reads better another way.

Layout direction

By default MindMap Studio balances branches on both sides of the root. For some maps a one-sided layout reads better -- an agenda or a process that flows top-to-bottom in your head often wants everything going **right** . The layout control in the toolbar switches between **both** , **right** , and **left** . It's a view setting; it changes nothing about the structure, so flip it freely to see which framing helps.

Boundaries: grouping a branch

Sometimes a branch needs to be visibly *a thing* : "everything in here is out of scope", "this cluster is Phase 1". Select the node at the top of the branch and click **Group** : a shaded, rounded **boundary** box is drawn around it and all its descendants, and double-clicking the box's label chip lets you name it. The boundary follows the branch as you edit it, so it keeps enclosing the right nodes even after you add or move children.

A boundary doesn't have to follow the tree, though. Select several topics -- from different branches, if that's the truth of it -- and **Insert -> Group selection (boundary)** draws one box around exactly those topics; double-click its label to name what they have in common. That's the box for the grouping the hierarchy can't express ("these five, wherever they live, are this quarter's bets").

For a *hierarchical* grouping -- one that lines up with a single branch -- the branch boundary above is the right tool. When the grouping is a genuine cross-cutting **link** between two ideas rather than a set, you want the next feature instead.

Relationships: the arrows across the tree

The tree is good at "X is part of Y". It is silent about "X depends on Y" when X and Y live on different branches. That's what **relationships** are for: hover a topic and a small **grip dot** appears on its edge -- pull it onto the other topic and the arrow is drawn (**Ctrl+Shift+L** starts one from the keyboard). Either way you've recorded a link the hierarchy couldn't express. A risk on the *Budget* branch that threatens a deliverable on the *Programme* branch; a decision that unblocks three others; a "see also". Use them sparingly -- a map laced with dozens of arrows is as hard to read as no structure at all -- but a handful of well-placed relationships often carry the most important information on the page.

> **Keep the tree honest.** Before drawing a relationship, ask whether the two nodes are really on different branches, or whether one should simply be re-parented under the other. An arrow is the right answer for genuine cross-links; it's the wrong answer for a hierarchy you just haven't tidied yet.

An arrow can also say *what kind* of link it is. Select a relationship and the inspector offers a semantic **type** -- *depends-on* , *causes* , *supports* , *blocks* , or the plain *relates-to* -- separate from its text label, plus one-click style **presets** (dashed, dotted, thick, curved, double-headed) when the line

itself should signal something. Types pay off later: the Power Filter and conditional rules can both ask "what blocks what?"

Once you have a few of these arrows, some of them will inevitably cross. Where two lines overlap, the eye can't tell whether they pass over each other or join -- and a false junction quietly tells the wrong story. Turn on **Line jumps** in the toolbar and the problem disappears: at every crossing, one of the two arrows lifts into a small **hop** over the other, the way a well-drawn wiring diagram or transit map keeps its lines legible. It's a per-map switch, so leave it off for sparse maps and flick it on the moment your arrows start to tangle; either way the hops are baked into your exports exactly as you see them on screen.

Floating topics

Not everything has to connect to the root. A **floating topic** is a node (or small sub-tree) that sits on the canvas unattached -- a parking lot for ideas you're not ready to place, a caption, a note-to-self. MindMap Studio renders floating topics imported from other tools in a labelled "Floating topics" branch.

A floating topic is a **first-class node** , not a second-class sticky: everything you do to a central topic, you can do here. Press **Tab** to add a child or **Enter** for a sibling; **Tab / Shift+Tab** indent and outdent (outdent a floating topic's child far enough and it becomes its own floating topic; indent one floating topic under another to nest them); right-click to **group it in a boundary** , **summarize** it, **paste a branch** beneath it, or **delete** it. Every inspector edit -- a note, a colour, task dates and progress, a hyperlink, markers and tags, a callout -- applies the same way. So a floating cluster is a genuine **staging area** : build it out as far as you like off to the side, then **drag it onto a branch** and the whole sub-tree joins the map. Drag a branch topic *out* and it floats free again.

Sticky notes

A floating topic is still a *topic* -- a node you might grow into a branch. Sometimes you want something lighter: a remark on the canvas that isn't part of the structure at all. The **Note** button drops a **sticky note** -- a free-floating amber card you can drag anywhere and type into. Amber is only the default: a swatch row beneath the Insert menu's **Sticky note** entry offers lime, sky, rose, violet, and slate, and the colour you pick becomes the new default -- so a convention like "rose = open question, lime = decision" survives from map to map. Use it for the things that hover *around* a map rather than *in* it: a "finish before Friday" to yourself, a caption over a cluster, a question for whoever you hand the map to. It's the canvas equivalent of a Post-it stuck to the whiteboard -- visible, movable, and plainly a note rather than a part of the tree. (For a remark pinned to *one specific node* , reach for a callout instead, Chapter 4; the sticky note is the free-floating kind.)

Summary topics

A boundary draws a box *around* a branch; a **summary** draws a labelled bracket *beside* one and says what it adds up to. Select the node at the top of a branch, click **Summary** , and a curly brace appears alongside it with an editable label -- "three options", "Q3 total", "needs sign-off". Where a boundary says *these belong together* , a summary says *here's the conclusion* . It's side-aware, so

on a two-sided map the bracket sits on the outer edge where it reads naturally, and double-clicking its label renames it. Reach for a summary when a branch has a punchline -- a total, a verdict, a next step -- you want on the page without adding another node.

Editing an overlay: the inspector

Boundaries, summaries, and callouts aren't just drawn-and-forgotten -- each is a thing you can come back to. **Click** any of them once and the right-hand panel switches to an **overlay inspector** for that object. There you rename it (the box's label, the bracket's caption, the callout's text), re-tint it from a small set of swatches -- the colour carries through to every export -- and, for a boundary, switch its shape (rounded, square, ellipse, cloud, polygon) and its outline (solid, dashed, dotted). A **Delete** button at the foot removes the overlay without touching the nodes underneath; the branch and its topics stay exactly where they were.

So if a Phase 1 boundary you drew earlier should now read as out-of-scope, click it, dash its outline and recolour it grey -- the grouping is the same, the signal is softer. And when a summary bracket has served its purpose, click it and delete it; the conclusion goes, the branch remains.

A vocabulary of shapes

By default every topic is a soft rectangle, and for most maps that's exactly right -- the *words* carry the meaning. But when a map is really a **process or a decision flow** , shape becomes meaning. Select a node, open the **Style tab** (in the Info panel), and you can recast it as a **diamond** (a decision -- "approved?"), an **oval** (a start or end point), a **parallelogram** (an input or output), a **hexagon** (a preparation step), or a **cylinder** (a data store). These are the classic flowchart shapes, and they read instantly to anyone who's seen one: a diamond *asks* , an oval *bookends* , a cylinder *stores* .

When the five classics aren't quite the picture in your head, the Style bar keeps going. A **trapezoid** marks a manual operation; an **octagon** says *stop* or *limit* ; a **document** -- a page with a softly waving bottom edge -- stands for a report or a printed output; a **callout** is a rounded speech bubble for an aside or an annotation; a **star** flags the one node that matters most; and a **cloud** is the universal shorthand for a loose idea or an external system you don't control. The concave ones (the star and the cloud especially) quietly pull their text inward so a long label never spills past the outline. Every one of these is drawn the same way on the canvas and in every export, so a flowchart you build here looks like a flowchart in the PNG, the PDF, and the Office decks. Use a shape when the geometry adds information; leave the soft rectangle when the word is enough.

Free-canvas mode

Auto-layout is a gift -- it keeps a growing tree tidy so you never nudge boxes around by hand. But some pictures aren't trees, and for those the tidy reflow gets in the way. Toggle **Free layout** and the auto-arranger steps aside: now you can **drag any node anywhere** and it stays put, its position saved with the map. This is the mode for a whiteboard-style diagram -- a system sketch, a seating plan, a freeform cluster of ideas -- where *where* a node sits is part of what it means. Switch Free layout off again and the tree snaps back to its automatic shape, every hand-placed position still

remembered for when you turn it back on. It's the escape hatch from the grid, not a replacement for it: most maps want the auto-layout, and the handful that don't want it badly.

Free placement comes with drafting aids. As you drag, **alignment guides** flash when a node lines up with a neighbour's edge or centre and the node snaps to them, so a hand-built layout still reads straight; **Align** and **Distribute** commands tidy a whole selection in one step. And a node you've placed deliberately can be pinned: right-click it and choose **Lock position** -- it gets a small lock badge, stops being draggable, and the tidy-up commands leave it alone.

Background shapes and smart containers

Sometimes the structure needs *scenery* : the four quadrants behind a SWOT, an arrow sweeping under a flow, lanes that split a plan by owner. **Insert -> Shapes** drops a free background shape behind your topics -- a **rectangle** , an **ellipse** , a **block arrow** , or a **chevron** -- sized and placed by hand (adding one switches the map to Free layout). Click a shape to move it, resize it from a corner, recolour it, or delete it from its small floating toolbar. Shapes are pure scenery: they sit behind the nodes, carry no children, and export exactly as drawn.

Two of the shapes are smarter. A **Swimlane** (a header band over vertical lanes) and a **Matrix** (a grid) are **containers** : any topic whose centre sits inside one belongs to it, and dragging the container **carries its topics along** in a single move -- slide the lane, its cards follow. There's nothing to enrol; membership is simply position. Between them, shapes and containers are how a mind-mapping canvas doubles as a whiteboard for the frameworks that are really *pictures* -- quadrants, pipelines, lanes.

Diagram backdrops: onion, funnel, Venn

Sometimes the *frame* carries the idea. The **Diagram** builder drops a geometric backdrop behind your topics for you to place them into: an **onion** of concentric rings (core to periphery -- values, then strategy, then tactics), a **funnel** of stacked stages narrowing to a point (a pipeline, a filtering process; **/+** changes the stage count), or a **Venn** of two or three overlapping circles (what's shared versus what's distinct). The backdrop is pure geometry -- it pairs naturally with Free layout above, since you're positioning topics into regions by hand -- and, like shapes, it's drawn identically on screen and in every export, so the diagram you build is the diagram you hand out.

A different layout for one branch

Chapter 5 covers the **Layout** dropdown, which re-flows the *whole* map into an org-chart, a timeline, a fishbone, and more. Occasionally a single *branch* wants a different shape from the rest -- an org-chart of the team hanging off an otherwise radial strategy map. Right-click the branch and choose **Branch layout** to give just that subtree its own arrangement. The override is sized as one block in the main layout so it never collides with its siblings, and overrides nest -- a branch inside the branch can have its own again. It's a precision tool; most maps never need it, but when one section is a fundamentally different kind of thing, this is how you let it look like one.

Now you try

Open your conference map (or any map with a few branches). Pick a leaf that sits under the wrong parent and **drag it** where it belongs -- watch the whole sub-branch travel and re-colour. Then choose one branch that's genuinely "a thing" and give it a **boundary** . Finally, find two nodes on *different* branches that depend on each other and draw a **relationship** arrow between them. Step back: the tree now carries the hierarchy, the boundary carries the grouping, and the arrow carries the one link the tree couldn't. If you can't find a real cross-branch link, that's a good sign -- don't invent one.

Two more moves worth a try. Flip the **layout direction** to right-only and back -- same structure, a different shape on the page; keep whichever frames the map better. Then drag a branch topic *out* into open space to make it a **floating topic** -- a parking lot for an idea you're not ready to place -- and drag it back onto a branch to re-attach it. Structure isn't a cage; it's something you reshape as the thinking firms up.

If your map is more diagram than tree, try the diagramming tools too. Cap a branch with a **Summary** that states its punchline. Turn a node into a **diamond** from the **Style bar** and watch the decision read at a glance. Or toggle **Free layout** and drag a few nodes into a shape you choose by hand -- then turn it off and watch the tree snap back, your positions remembered. None of these replace the tree; they're there for the maps the tree alone can't draw.

Three more moves for the maps that need them. Right-click a single branch and choose **Branch layout** -- give just that sub-tree an org-chart while the rest of the map stays radial; the layouts compose, so one busy branch can read its own way without reshaping anything else. When a map is really a framework, open the **Diagram builder** and drop topics into an onion, a funnel, or a Venn backdrop -- the frame carries the meaning so the labels don't have to. And reach past the diamond: the shape menu also holds a trapezoid, octagon, document, star, and cloud, each drawn identically on the canvas and in every export. One last doorway -- when two far-apart topics refer to each other but aren't a true dependency, give one a **jump link** to the other; clicking it flies the canvas to the target, the connection without the arrow. (You can also drop a web link or a block of text straight onto empty canvas and watch it land as a floating topic, ready to place.)

If you drew a boundary or a summary above, **click it once** now: the inspector opens on the right. Rename it, pick a colour, and -- for a boundary -- try the dashed outline. Then **delete** one and watch the branch underneath stay put; the grouping was never holding the nodes, only enclosing them. Now draw a second **relationship** arrow that crosses the first, then turn on **Line jumps** in the toolbar. Where the two lines met as an ambiguous junction, one now lifts in a small hop over the other and the crossing reads cleanly. Toggle it off and the hop vanishes -- it's a per-map switch, so save it for the moment your arrows start to tangle. Either way, export the map and the hops come through exactly as you see them.

Two quick reuse moves to finish: **right-click a branch -> Copy branch** , then paste it under another node -- or even into a *different* map -- and notice the copy is independent of the original. And drop a **sticky note** somewhere with a reminder to yourself -- picking a non-default colour from the swatch row while you're at it -- a remark that rides *with* the map without becoming part of its tree.

And two ordering moves. Select a topic with a handful of children and run **Sort children A -> Z** from the palette -- then **Ctrl+Z** it, because alphabetical was probably not the meaning. Then lasso

topics from two *different* branches and **Insert -> Group selection (boundary)** : a grouping the tree itself couldn't draw, named with a double-click on its label.

With structure under control, the next chapter makes individual nodes carry more.

Enriching nodes

Notes, markers, images, and making a map look like it means it

A bare tree of labels is often enough. But a node can carry far more than its topic, and used with restraint, that extra detail turns a sketch into a document you can hand to someone else.

Notes: the prose behind the bubble

A topic should stay short. The paragraph it deserves goes in a **note** . Select a node, open the **Notes** panel, and write. Notes accept **Markdown** -- headings, bold and italic, bullet lists, inline `code` , and links -- and the panel shows a live preview, so the formatting you type is the formatting you get.

Notes are where a map stops being a brainstorm and becomes a brief. The node says *Vendor decision* ; the note records the three options, the criteria, and why you chose the one you did. The canvas stays scannable; the detail is one click away on whichever node owns it.

Less than a click, in fact. A node with a note grows a small indicator, and merely *hovering* it peeks the note in a small card -- enough to check a detail without opening the panel; clicking it opens the editor. The same trick works elsewhere: hover a collapsed branch's **+N** count and it peeks the first few hidden children, and a node's attachment chip lists its file names. The map keeps its depth behind the labels, but the depth answers to a glance.

Markers: status at a glance

A **marker** is a small icon or tag pinned to a node. Open the **Markers** palette and click to toggle one on the selected node: a priority flag, a tick, a question mark, a face. Each chip also **drags** -- pull a marker from the palette onto any topic on the canvas, selected or not, and it lands where you drop it, which is the natural motion when you're triaging a whole map rather than dressing one node. Markers are how a map carries *state* without words -- a row of green ticks and one red flag tells a reviewer where to look before they've read a single label.

Pick a small vocabulary and stick to it. "Red flag means blocked, tick means done, question mark means undecided" is a convention a whole team can read at a glance; fifteen different icons used once each is just clutter.

The palette shows a curated starter set; a **search box** above it reaches the full library of forty-five by name ("done", "warning", "idea"). Four of the sets are **single-select groups** -- Priority (1-9), Status (coloured dots), Mood, and Vote -- so applying one member clears any other from the same group and a topic never carries two priorities at once. And the app makes small suggestions of its own: a **Suggested** row under the palette offers markers inferred from the topic's text -- write "urgent" and the is one click away. The suggestions never apply themselves; the vocabulary stays yours.

Tags get housekeeping to match. The tag field **autocompletes** from every tag already in the map, and the **Markers & tags** index panel can **rename** a tag everywhere, **merge** two by renaming one

onto the other, **delete** one from every topic, or give it a **colour** that tints every topic carrying it -- a view-only wash that a node's own styling still overrides. Fastest of all, don't visit a panel at all: type # while editing a topic and the tag picker pops up mid-keystroke -- pick an existing tag or mint a new one, and keep typing.

> Markers imported from a MindManager `.mmap` file are mapped to the closest emoji, so a map you bring in from elsewhere keeps its visual cues rather than arriving as bare text.

When you select several nodes at once, the Markers palette switches to **bulk** mode and works across the whole selection. A marker chip reads its state from the group: it shows **lit** when every selected topic already carries that marker, and **dashed** when only some of them do. Clicking follows the obvious rule -- a lit chip *removes* the marker from all the nodes; a dashed or empty chip *adds* it to every node that lacks it, in a single undo step. So to flag a dozen blocked topics, lasso them, click the flag once, and it lands on all twelve; click it again and it's gone from all twelve. Tags behave the same way in bulk: select the cluster, type or click a tag, and `risk` is on every node in the group at once -- which is exactly what a conditional rule (below) is waiting for.

Tasks: when a topic is also a job

A marker says *this is urgent* ; a **task** says *this is work, and here's where it stands* . Open the **Info** panel and a node can carry three task fields: a **progress** slider (0 to 100%), a **priority** (the full 1-9 ladder MindManager uses -- 1 reads as High, 2 as Med, 3 as Low, and 4-9 as their number), and **start** and **due** dates -- fields that understand plain language, so `tomorrow` , `next fri` , or `+7d` resolve to real dates as you leave the field. Together they turn a branch of a plan into something you can actually track.

Progress **rolls up** : set the leaves and a parent shows the average of its children, so the top of a branch reports how far the whole thing has come without your doing the sum. Due dates work the same way -- a topic past its date is flagged **overdue** on the canvas, so a plan that's slipping says so out loud. None of this turns MindMap Studio into a project planner; it makes a *map* enough of one that you don't have to leave it to see what's done, what's next, and what's late. (Chapter 5's filter and board then let you ask the whole map "what's overdue?" at once.)

Day to day you rarely need the panel. Hover a topic's left edge and a small **checkbox** appears: click it to cycle *not a task -> to-do -> done* -- the fastest way to run a checklist straight on the canvas. And when a plan slips wholesale, don't edit dates one at a time: **right-click** the branch's top node and the **Shift task dates** row moves every start and due date beneath it by **-1w / -1d / +1d / +1w / +1mo** in one undo step, relative gaps preserved. The deadline moved a week; the map takes one click to agree.

Images

A node can hold an **image** -- a screenshot, a logo, a photographed whiteboard, a chart. Attach one to the selected node and it renders inline on the canvas -- and getting one in is as direct as **Ctrl+V** : copy a screenshot anywhere and paste it straight onto the selected topic. An image is worth a paragraph of note when the thing you're describing is itself visual.

An image usually sits *beside* the label. For a stronger statement, the Style tab's **Fill image...** puts a picture *behind* it instead -- covering the whole card, the topic text over it, one per node, with **Clear fill image** to lift it off. It reads like a small poster: right for the handful of nodes that *are* their picture (a person, a place, a product), wrong for twenty.

Sometimes you don't have a file to hand -- you just want a small visual cue: a **star** on the idea you're proudest of, a **warning** triangle on the risk, a **flag** on the decision still open. For that, the **Info** panel keeps a built-in grid of **stickers** : twenty clean little illustrations -- star, heart, check and cross badges, lightbulb, target, rocket, lock, clock, fire, and more -- in one quiet accent colour so they read as a set rather than a circus. Click one and it lands on the selected node. Behind the scenes a sticker simply *becomes* that node's image, which is the whole trick: it renders on the canvas and travels into every export exactly like a picture you supplied yourself, with nothing new to learn. A node holds one image at a time, so picking a new sticker (or a real photo) swaps out the last. Used sparingly -- one or two on the nodes that carry the most weight -- a sticker pulls the eye to what matters without a word.

Attachments: files that travel with the node

An image renders *on* the node; an **attachment** rides along *with* it. From the **Info** panel you can attach a file to the selected topic -- a spec, a contract, a spreadsheet -- and it's stored inside the map itself, so it travels in the JSON export and is there even offline, one click from downloading again. Reach for an attachment when the source document matters but doesn't belong on the canvas: the node says *Vendor contract* , the file *is* the contract, kept with the idea it backs up.

Styling: shape, fill, border, font

Beyond markers, individual nodes can be **styled** : change a node's shape, fill colour, border, or weight; bump a topic's font size or colour; and switch its **font family** from the **Font** picker -- the default sans-serif, a **serif** for a quieter, document-like topic, or **monospace** for code, IDs, and anything that should line up. Two fill options go beyond a flat swatch: a **branch-colour tint** () washes the node in a soft version of its own branch's colour -- structure restated as colour, no palette decisions required -- and a **gradient** () adds a gentle fade for the one node that should read as a header. And a topic that has outgrown its line doesn't have to stretch the layout: set its **wrap width** -- the snap slider in the Style tab, or the width grip on the selected node itself -- and long text folds into a tidy block instead of a banner. Styling earns its keep when it *encodes* something -- the three "decision" nodes all share a fill, the headline branch is bolder than the rest -- and costs you readability when it's merely decorative. Style to create a pattern the reader can rely on, not to make the map "pop".

Styles you can reuse, formatting that follows the data

Hand-styling one node at a time is a scalpel; two features in the **Styles** panel (the toolbar button, not the per-node Style bar above) make styling *systematic* .

A **named style** saves a look so you can reuse it. Dress one node the way you want -- fill, border, font, weight -- save it under a name, and from then on a click applies that exact look to any selected

topic, in this map or any other. "Our decision nodes look like *this* " stops being something you redo by hand; the styles you save are kept app-wide, so a visual language you settle on follows you across the whole library.

Conditional formatting goes further and styles nodes *by what they are* . Write a rule -- "anything tagged `risk` gets a red border", "anything 100% complete goes grey", "every node with a marker turns amber" -- and the map applies it everywhere, live, and keeps applying it as you edit. The triggers reach well past tags: a rule can fire on a marker, on completion, on being **overdue** or **due soon** , on a **priority** threshold, on the topic's **text** , on carrying an **attachment** , or on having a **relationship** of a given type -- and conditions can be chained with **AND** and inverted with **NOT** ("tagged `q3` and *not* completed"). A rule can also apply a **marker** or a **branch colour** , not just a fill. Where a named style is a look you *apply* , a conditional rule is a look that *follows the data* : tag a new node `risk` and it goes red without your touching the Style bar. On a big, changing map that's the difference between formatting you maintain and formatting that maintains itself.

Two smaller tools round out the systematic kit. The **Format painter** copies one topic's look and pastes it onto any selection -- reuse without the ceremony of naming a style. And **Auto-colour branches** repaints every top-level branch a distinct colour from the current theme's palette in one click, the fastest way to give a grown map a legible colour structure.

Rich text: emphasis inside a topic

Styling paints the whole node; sometimes you want to stress just a word or two *inside* the label. While editing a topic, **Ctrl+B** , **Ctrl+I** , and **Ctrl+U** bold, italicise, or underline the selection -- the same muscle memory as any editor -- and a small **floating bar** above the topic offers the same three plus a row of **text colours** for the mouse path. Reach for it sparingly: one bold word that names the decision, an italicised term you're defining. The plain text is kept underneath, so your outline and every flat export stay clean even when the canvas is dressed up. (A **Spell-check** toggle in the View menu turns the editors' squiggles on or off map-wide -- off is a mercy on a map full of product codenames.)

Callouts: a note that points

A **callout** is a small sticky note pinned beside a node -- a caveat, an open question, a "revisit this" -- without promoting it to a child topic. Right-click a node, choose **Add callout** , then double-click the bubble to write in it. Unlike a note, which lives *behind* the node one click away, a callout sits *on the canvas* , visible at a glance and drawn into your image exports -- the right tool for the one remark a reader must not miss. Like markers, their power is in scarcity: a map speckled with callouts has none.

A boundary, a summary bracket, and a callout each carry their own **colour** . Select one and the inspector offers a small row of swatches plus a **Default** button; the colour you pick re-tints the whole object -- its outline, fill, and label together -- on the canvas and in every export, and **Default** drops it back to the theme's accent. Use it the way you use everything else here: to *encode* . Tint the "out of scope" boundary grey and the "this release" one green and a reader sees the split before reading a word; leave the rest on the default accent so the two coloured ones carry the meaning.

Themes: the whole canvas at once

Where per-node styling is a scalpel, a **theme** is a coat of paint for the entire map. The theme gallery offers a few presets -- **Light** , **Dark** , **Ocean** , **Sunset** -- each a coordinated palette for the background, branches and text. Dark themes read well on a projector in a dim room; light themes print cleanly. Switching theme never touches your content, so try a few and keep whichever helps you see the map. When the presets aren't yours, **Manage themes...** opens a small designer: name a theme, pick its six-colour branch palette, background, font and branch weight, and it joins the gallery -- exportable as a file, so a house style travels between machines. (The canvas theme is separate from the *app's* chrome, which follows your operating system's light/dark setting -- or force it in Settings.)

One level up again, the Map panel's **Design gallery** applies a whole coordinated look -- theme, connector style, branch weight, accent -- in one click: the "make it presentable" button for when the content is done and the meeting is soon. The same panel's **More styling** section sets the map's **typography** : a base **font** (Sans, Serif, or Mono) and a **text size** (Compact, Comfortable, Large) for the whole map, which any per-topic font or size still overrides. A serif base reads like a document; Compact buys a large plan another dozen visible topics.

The **Canvas** colour control sits one notch below a theme: it overrides just the background of *this one map* , leaving the theme's branch and text palette intact. It's a quiet but useful signal when you keep many maps -- a faint green wash on the "ideas" map, a warm one on "risks" -- so you know which map you're in at a glance. The colour saves with the map and follows it into an image or PDF export; the clears it back to the theme.

Right beside it, the button goes a step further and lets you drop a whole **image** behind the map -- a faint grid, a watermark, a photograph of the whiteboard you're rebuilding, your team's brand backdrop. Pick a file and it fills the canvas behind every topic, sitting on top of the background colour (so a transparent PNG lets that colour glow through the gaps). The picture is shrunk to a sane size and tucked inside the map itself, so the map stays a single portable file you can open offline -- and, like everything else here, what you see is what you get: the backdrop travels into your SVG, PNG, PDF and HTML exports unchanged. Use it sparingly, though; a busy photo behind your topics fights the words for attention, and the map is there for the words. The second lifts the image back off.

A note on restraint

Every feature in this chapter can be overused. The test is always the same: *does this make the map easier to think with?* A note that captures a decision -- yes. A marker convention a team shares -- yes. Six fonts and a gradient on every node -- no. The most useful maps are usually the plainest ones with enrichment applied exactly where it carries meaning.

One quiet piece of bookkeeping needs no decision from you at all. Select a node and the inspector header shows, in faint text under the breadcrumb, when the topic was **created** and last **modified** -- "created 3 d ago, modified 2 h ago", and a plain date once a change is more than a week old. You never set these; the map keeps them. They earn their place when a map outlives the meeting that made it: glancing at a branch and seeing it hasn't been touched in a month tells you whether it's

settled or stale before you reopen the argument.

Now you try

Take a node that's carrying too much in its label and move the detail into a **note** (open the Notes panel and write a sentence or two in Markdown). Shorten the topic to a handful of words. Then agree a tiny **marker** convention with yourself -- say, a flag for "blocked" and a tick for "done" -- and apply it to three or four nodes. Read the map again: the canvas should be more scannable than before, with the depth one click away. If you reach for a fifth marker colour, stop -- the small vocabulary is the point.

Now add a layer of meaning. Attach an **image** to one node -- a screenshot or a logo -- and watch it render inline. Pick three related nodes (say, the ones that represent decisions) and give them a shared **fill** from the Style bar so the pattern reads at a glance; bump your headline branch's **font** up a size so the eye starts there. Finally, open the **theme** gallery and switch between Light and Dark -- your content doesn't change, only its coat of paint, so keep whichever helps you see the map. The rule throughout: styling that *encodes* something earns its place; styling that merely decorates doesn't.

Finally, dress one topic up *inside* its label -- bold the single word that names what it is (**Ctrl+B** while editing) -- and pin a **callout** to the node you're least sure about, holding the question you still need to answer. Notice how differently the two read: the bold word is part of the idea; the callout hovers beside it, plainly a comment on the work rather than the work itself.

Finally, make a node carry *work* , not just words. In the **Info** panel give a topic a **due date** and drag its **progress** to half-done -- watch the parent branch show the rolled-up total and the canvas flag anything overdue. Then open the **Styles** panel, write one **conditional rule** (tag a node `risk` , have the rule paint it red), and add that tag to a second node: it turns red on its own. That's the chapter in miniature -- a node that tracks its own status, and formatting that follows the data instead of waiting for your hand.

Two more passes turn a handsome map into a working one. Give a topic a **priority** -- 1 for High, down to 9 for the barely-urgent -- then switch on the priority filter and watch everything below your cut-off fade; that is triage in two clicks. Pick a font that suits the map's register -- the Styles bar offers **Sans, Serif, or Mono** per topic -- and, once a node looks exactly right, **save its look as a named style** and drop that style onto its siblings so the set moves as one. Hang a real **file** off a topic -- a spec, a slide deck -- and it travels inside the map, downloadable whenever you reopen it. Reach into the **sticker** library when a small picture says more than a label would. And if the canvas itself wants a mood, give the whole map a **background colour** , or a faint background **image** behind everything -- it renders the same in every export, so what you arrange is what you ship.

One more pass, this time on a group. Select three or four nodes at once -- the Markers palette switches to **bulk** mode. Click a marker and watch it land on the whole selection; notice that a chip already on *all* of them reads **lit** (click to clear all) while one on only *some* reads **dashed** (click to add to all). Add a **tag** across the same selection the same way. Then pin a **callout** and open its inspector: give it a **colour** from the swatch row so the remark reads as deliberately set-apart, and try **Default** to drop it back. Finally, just look at a node's inspector header -- the faint **created** and

modified line is there without your having lifted a finger, and on an old map that one glance tells you what's still alive and what has gone quiet.

A last pass for this chapter's quieter tools, in one loop. Hover a noted topic's and peek the note without opening anything; **drag** a marker from the palette onto a topic you haven't even selected; give one node a **branch-colour tint** and set a **wrap width** on your longest label so it folds instead of stretching. Then run a checklist from the canvas: hover three topics' left edges and click the **checkbox** through to-do and done. Finally, right-click your plan's top node and **shift its task dates +1w** -- the whole branch slips together, gaps preserved -- and **Ctrl+Z** the slip away again. None of these needed a panel; the canvas was the interface.

The next chapter is about the opposite problem: when a map gets big, how do you keep finding your way around it?

Navigating large maps

Outline, find, collapse, and links between maps

A map with thirty nodes fits on a screen. A map with three hundred does not, and the features that felt optional at small scale become the difference between a map you use and a map you abandon. This chapter is about staying oriented.

Collapse and expand

Every branch can be **collapsed** to hide its children behind its parent, and expanded again to reveal them. Collapsing is how you control altitude: fold everything down to the top two levels to see the shape of the whole plan, then open just the branch you're working in. The toolbar's **collapse all** and **expand all** controls do it to the whole map at once -- collapse-all then open one branch is the fastest way to present a large map without overwhelming the room.

Three refinements make altitude precise. **Detail levels** ("Show level 1", "level 2", ...) expand the map to an exact depth in one command -- the step-wise version of collapse-all. **Isolate branch** collapses every *other* top branch and opens the path to the topic you're on: "fold everything else away" as a single, undoable move. And **Drill in** goes further than folding -- it *re-roots* the view at the selected topic so its subtree fills the screen; you keep editing as usual (it's just a view), and a bar or **Esc** brings the whole map back. While you're deep in a branch, a **breadcrumb** above the canvas shows the path from the root -- click any ancestor to climb back up without scrolling.

Fit

Lost? The **Fit** button reframes the entire map to the viewport in one click. It's the "take me home" of the canvas, and worth wiring into muscle memory: zoom in to work, hit Fit to see where you are.

The minimap and zoom

In the bottom-right corner sits a **minimap** -- a shrunk-down overview of the whole map, with a highlighted rectangle showing the slice you're currently looking at. On a big map it answers the question "where am I, and what else is out there?" at a glance. Click or drag inside it to jump the main view somewhere else: the rectangle follows your pointer and the canvas pans to match, so the minimap doubles as a fast way to travel across a map too large to scroll comfortably.

Below it are the **zoom controls** -- minus, a live percentage, plus, and a fit button. They do the same job as the mouse wheel but give you a precise readout and a deliberate step, which matters when you're lining a map up for a screenshot or a screen-share. The percentage tells you exactly how far in you are; the fit button is the same "take me home" as **Fit** above, kept within thumb's reach of the zoom buttons.

Layouts: the same map, different shape

The map's default shape -- a two-sided radial -- isn't the only way to read it. The **Layout** dropdown

re-flows the *same* nodes into a different arrangement, and which one reads best depends on what the map *is* :

- An **org-chart** (top-down or bottom-up) suits a hierarchy: a team, a taxonomy, a decomposition.
- A **timeline** lays the first level out as a left-to-right sequence -- a roadmap, a process, the steps of an argument.
- A **fishbone** (the Ishikawa diagram) angles branches into a spine, the classic shape for cause-and-effect analysis.
- **Radial** fans every branch evenly around the root when you want the pure, balanced bloom.
- A **grid / matrix** tiles the top branches into cells -- four branches make a 2x2, the natural shape of a SWOT or an Eisenhower box.
- A **swimlane** runs the top branches as parallel lanes -- a process split by owner or stage.
- A **brace map** joins each parent to its children with a { fork -- the part-whole diagram, read left to right.

The picker itself is a **gallery** : each layout shows as a small schematic thumbnail in the Map panel, so you choose by shape, not by name.

Switching layout never changes your content -- only its geometry -- so it costs nothing to try a few and keep the one that makes the structure obvious. A backlog that felt tangled as a radial map can read as a clean timeline; a muddled list of problems snaps into focus as a fishbone.

The outline panel

The **Outline** panel shows your map as an indented list -- the same tree, read top to bottom instead of radiating outward. It's invaluable for three things: seeing the whole structure linearly, jumping straight to a node (click it in the outline and the canvas focuses it), and spotting structural problems a radial layout can hide, like a branch that's gone six levels deep while its siblings stayed shallow.

It's also a full **editor** , not just a table of contents. **Double-click** a row to rename it inline -- and while you're editing, **Tab** adds a child and **Enter** a sibling, exactly like the canvas. The buttons on a row promote and demote it; **drag** a row to reorder or re-parent (on a touchscreen, hold a row a beat and slide it); **Shift+Up / Down** reorder among siblings and **Shift+Left / Right** promote and demote from the keyboard, focus staying on the moved row. A long restructuring session is often *faster* here than on the canvas -- linear is the better shape for triage -- and the two views stay in step both ways: select a node on the canvas and the outline scrolls its row into view.

The outline has its own **filter** box: type a few letters and it narrows to matching topics, so a long map becomes a short list you can scan.

The marker and tag index

The outline answers "where is this *topic* ?" The **Index** panel (the **Index** button) answers a different question: "where is everything I *flagged* ?" It collects every marker and tag used anywhere in the map and lists them grouped by symbol -- a heading with the three topics you marked urgent under it, a heading with your favourites, and so on, each with a count. Click a topic in the index and the canvas jumps to it, exactly like the outline.

This is the read-only twin of the **Markers** palette. The palette is how you *put* a marker on the selected node; the index is how you *find* every node that already carries one. On a map with a hundred topics, that's the difference between scanning the whole thing for red flags and reading them off a list. Each row also offers a **Quick Filter** : one click and the canvas shows just the topics carrying that marker or tag -- the flag-then-ask loop closed in a single gesture. Markers earn their keep precisely because the index makes them queryable after the fact -- so flag deliberately, and the index becomes a live status board.

Numbering the branches

A radial map is wonderful for thinking and terrible for *pointing* . "Look at the third sub-point of the second branch" is a sentence no one wants to say. The **1. Numbering** toggle fixes that: it stamps every topic with its outline number -- `1` , then `1.1` , `1.2` , then `1.2.1` -- on the canvas and down the outline at once, so now you can just say "see 2.3" and everyone's eyes land in the same place. The numbers follow into the PNG, PDF, and Office exports too, which is exactly when you need them: a printed map handed round a meeting, or a slide that references a node by number.

It's worth being clear about what numbering *isn't* . It doesn't change your map. The numbers are computed from the tree's shape the instant you switch it on and forgotten the instant you switch it off -- they never touch your topic text, so search still finds "Budget" and not "4.2 Budget". Reorder a branch and the numbers renumber themselves, because they were never really yours to begin with -- they belong to the structure. They do follow you *out* , though, and further than the images: while numbering is on, **Copy outline** and the **Markdown export** bake the numbers into each line too, so the reference you say out loud ("see 2.3") is the same one in the minutes; switch numbering off and the same exports come out clean. A **Numbering style** item under the toggle picks the scheme -- **decimal** (1, 1.1) or **outline** (I, A, 1) -- so the map can match the document it's headed into. Turn it on when you need to talk *about* the map; turn it off when you just want to look *at* it.

The legend: your conventions, on the page

Chapter 4's advice was to keep a small marker vocabulary -- but a reader who wasn't there when you agreed it still needs the key. Toggle **Legend** (in the View menu, under Display) and a quiet box appears in the map's corner listing exactly the conventions *this map actually uses* : each marker with its name, each coloured tag, each conditional-formatting rule with its colour. Nothing in use, nothing listed. It's drawn on the canvas, so it rides into the SVG and image exports unchanged -- the handed-out PNG explains its own symbols. Turn it on for maps that travel; leave it off while the only reader is you.

Filtering without losing the forest

There's a moment with every large map where you stop wanting to *see everything* and start wanting to *see one thing* : every red flag, every node tagged `q3` , every topic that mentions "budget". The **Filter** panel does that without throwing anything away. Type some text, or click the marker and tag chips for what you're hunting, and the map answers by **fading everything that doesn't match** -- leaving the matches, and the branches that lead to them, at full strength. A count tells you how many topics qualified.

The phrase to hold onto is *read-only* . A filter here is a spotlight, not a pair of scissors: by default everything else is only dimmed, and the instant you hit **Clear** or close the panel the whole map comes back. When even the dimmed context is noise -- a screenshot meant for one team, a map too dense to read faded -- tick **Hide non-matches** and the rest leaves the canvas entirely: still the same spotlight, the map itself untouched (the panel's footer says so), just with the house lights fully off. That read-only rule is deliberate. The fastest way to lose trust in a tool is to have it quietly remove something you needed; a filter that never edits the map -- only what's shown of it -- can't. The context stays too -- because the path from the centre to each match keeps its colour, you never get the disorienting "lone highlighted node floating in grey" that makes you forget where you are.

It pairs naturally with markers. Flag the risks as you build the map (Chapter 4), then, when the map is big and the meeting is short, filter to and read the risks straight off the lit branches. The markers are the *input* ; the filter is the *question you ask of them later* .

A filter you build often, you shouldn't have to rebuild. **Save** one as a named **preset** and it joins a list you re-apply in a click -- " risks", "due this week", "tagged `q3` " -- and because presets are kept app-wide, a filter that proves useful on one map is waiting on the next. The Power Filter stops being something you set up each time and becomes a set of saved questions you can ask of any map.

Two more controls round it out. A **completion** selector -- *Done* , *In progress* , *Not done* -- makes "what's genuinely finished?" a click instead of an audit. And when a filtered view turns out to be a *thing in its own right* -- the risk register hiding inside the programme map -- **Extract matches to a new map** copies the matching topics, with the branches that lead to them, into a fresh map of their own and leaves the original untouched: the spotlight, promoted to a document.

The filter answers "where is everything matching X?" Its close cousin, **Focus** , answers "show me just *this* ." Select a node, click Focus, and the map dims to that node's branch and the single path back to the centre -- the same spotlight-not-scissors idea, aimed at one branch instead of a query. It's how you walk a meeting through section 3 of a forty-node map without the other thirty-odd nodes competing for attention; **Esc** brings them back. Reach for Focus when you know *which* branch you mean, and the filter when you're asking the map a question across all of them.

Board view: the map as columns

A map is a tree; sometimes the question you have is a *board* question -- "what's in each bucket?" The **Board** button answers it without duplicating your map. It lays the topics out as **Kanban columns** , grouped by whichever source you pick in its header: **tags** (one column per tag -- `todo` / `doing` / `done` , or `backlog` / `now` / `next`), a single-select **marker group** (Priority, Status, Mood, Vote), or the **schedule** (Overdue / Today / This week / Later). Each card shows the topic with its rolled-up completion and due date.

The board is a *lens on the same map* , and it writes back through the lens: **drag a card to another column** and the topic itself is re-tagged, re-marked, or re-scheduled -- the tree is updated, not a copy. Click a card and the canvas jumps to that topic. Like the filter and the index, it's the same data asked a different question: the tree is how the work is *organised* , the board is how it's *progressing* -- except here the answer is also a control.

Find and replace

Press **/** anywhere to jump straight to **Find** . It searches everything a topic carries -- its text, its note, its tags and markers, even attachment names and links -- so a term you only mentioned in a note is still findable. Matches are highlighted and you can step between them, or open **List all** for a clickable list with each match's breadcrumb. Find is also **forgiving** : if what you typed matches nothing exactly, it quietly falls back to a typo-tolerant pass, so a half-remembered spelling -- *recieve* for *receive* -- still lands you on the node instead of an empty result. **Replace** turns find into a tidy-up tool: rename a project that got called three slightly different things, fix a term you decided to change, all in one pass. Two conveniences ride along: the Find box remembers your **last ten queries** (start typing and they're offered back), and Replace has a **scope** -- *Topics* , *Notes* , or *Both* -- so renaming a project everywhere doesn't have to mean rewriting the prose behind the nodes, or vice versa.

When the question is sharper than a word, the same box speaks **operators** : `tag:q3` , `marker:flag-red` , `priority:1` , `due:overdue` , `due:soon` , `has:note` , `has:attachment` , `level:>=2` , `-exclude` , `"exact phrase"` -- combinable, and understood by the across-maps search too. "Every overdue topic tagged `q3` that has a note" stops being a scan and becomes a query.

> The `/` -to-find shortcut is the single most useful key on a big map. When you can't remember where something is, don't hunt -- press `/` and type.

The command palette: do anything by name

Find locates a *topic* ; the **command palette** locates an *action* . Press **Ctrl/Cmd + K** anywhere in the editor and a single search box opens over the map. Type a few letters of what you want -- `fit` , `collapse` , `board` , `outline` , `timeline` , `export pdf` -- and the list narrows to matching commands; arrow down to one and press **Enter** to run it. Every button on the toolbar is in here, so you never have to remember which row or menu hides a control: you just name it. The match is *fuzzy* -- it accepts a subsequence, so `cllpse` still finds **Collapse all branches** and `xpdf` still finds **Export .pdf** -- and the palette remembers your last few choices under a **Recent** heading, so the things you do often are one keystroke and Enter away the next time you open it.

The palette is selection-aware. Select a node first and a band of commands appears that act on *it* -- add a child, set a marker or priority, focus its branch, delete it -- so a node's whole context menu is reachable by name without touching the mouse. With nothing selected, those commands quietly drop off the list rather than offering you an action with no target.

Jump to any topic from the palette

The same **Ctrl/Cmd + K** box is also the fastest way to *travel* . Alongside the actions, every topic in the map sits in the list as **Go to: <topic>** ; type part of its text, press Enter, and the canvas selects and centres that node. The trick that makes it worth the keystroke is what it searches: each topic's row quietly folds in its **note text** as well, so a term you only wrote in a note -- never in a topic title -- still surfaces the right node. It is the `/` -to-Find idea widened to the whole map at once: one box that finds both the thing you want to *do* and the place you want to *be* . The list even reaches past the

current map: every other map in your library sits there as a **Switch to map:** row, so changing documents is a few keystrokes, not a trip to the dropdown.

The keyboard shortcuts cheat-sheet

When you forget a binding, you don't have to leave the map to look it up. Click the **? (help)** button in the icon rail -- or open the command palette and run **Keyboard shortcuts** -- and a cheat-sheet lists every editing, navigation, and view shortcut: Tab for a child, Enter for a sibling, `/` for Find, **Ctrl/Cmd + K** for the palette itself, and the pan-and-zoom gestures. The sheet is generated from the same shortcut table the app actually binds, so what it shows is always what the keys really do. Skim it once and the moves in this chapter stop being things you memorise and start being things you reach for.

Keyboard-reachable menus, and the same map on a phone

None of this assumes a mouse. The toolbar's dropdown menus and a node's right-click context menu are fully keyboard-driven: open one and the arrow keys walk the items, Home and End jump to the ends, Enter runs the highlighted one, and Escape closes without choosing. On a narrow screen the layout adapts rather than breaking: the side panels -- Outline, Index, Filter, Info -- slide up as a **bottom sheet** over a full-width canvas instead of crushing it into a sliver, so the same map you build at a desk stays usable on a phone.

Links: doorways between topics and maps

A radial map is a tree, but real subjects aren't: the risk you noted on one branch is the same risk that constrains a plan three branches away. Drawing a relationship line between them is one answer (Chapter 3); a **link** is the lighter one. Give a node a link and it grows a small -- click it and you travel.

A link can point three ways, all set from the **Info** panel. **Jump to a topic** points it at another *topic in the same map* , so "see also: Budget" becomes one click instead of a hunt -- the canvas leaps to that topic and selects it, no line cluttering the picture. **Link to a map** points it at *another map* in your library (Chapter 7): a node in your *Strategy* map can open your *Q3 Plan* map -- and, with the topic picker that appears once a map link is set, land on a *specific topic* inside it. That's how you build a small **atlas** instead of one unreadable continent -- a high-level map whose nodes are doorways into the detailed maps beneath them, with a **Linked from other maps** section in the inspector showing the doorways that point back. And a plain web URL points it at a page, which opens in a new tab. The carries one primary link; further URLs live in an **Additional links** list on the same topic. Fastest of all, while *typing* a topic: `[[` or `@` autocompletes any topic or map by name and attaches the link as you pick it.

The inspector keeps the ledger for a single topic: alongside **Linked from other maps** (the doorways pointing in), a **Links to** section lists the outgoing side -- the topic's own hyperlink plus every relationship leaving it, click-to-jump. And placed one by one, the whole map's connections can be read back all at once: the **Relationships** panel (Panels menu, under Analysis) lists every relationship arrow and every in-map topic link, both ends click-to-jump, labels shown. On a map where the arrows carry the real story -- a dependency web, a risk map -- the panel *is* the story, in

rows; it's also the fastest way to find the arrow you meant to delete.

Its neighbour in the Analysis section, **Map statistics** , is the map's vital signs: topics, leaves and depth; task counts with done-so-far and overdue; the content tally -- words, notes, attachments, tags, relationships -- and a **reading time** estimate. The number worth watching is the word count: when a "map" passes a few thousand words it has usually become a document wearing a map's clothes (Chapter 2 said when not to map; the statistics panel is where you catch it happening).

Search every map

Find searches the map you're in. Once your library grows into an atlas, **All maps** searches *all* of them at once -- every topic and note, floating topics included -- and picking a result opens that map and lands on the node. It's the companion to cross-map links: links are the doorways you placed on purpose, library search is for when you can't remember which map a thing is in.

Roll-ups: a node that mirrors another map

Cross-map links are *doorways* -- you click through to the other map. A **roll-up** brings the other map's content *to you* . Select a node, pick a source map from the **Roll-up** menu, and that node becomes a live mirror of the source: the source's top-level branches are copied in beneath it. Click **Roll-ups** whenever you like and every roll-up node re-pulls the latest from its source, so a high-level map can *aggregate* the maps below it instead of merely pointing at them.

This is the automated cousin of copy-paste (Chapter 3): a paste is a one-time snapshot you then own and edit; a roll-up stays **bound** to its source and refreshes on command. It's how you build a genuine **dashboard map** -- one overview whose branches are the current state of half a dozen detail maps -- and keep it current with a click rather than re-copying by hand. Treat a roll-up node as a window onto the source, not a place to edit: the mirrored branches are refreshed wholesale, so do the editing in the source map and roll it up again.

Retracing your steps, and views worth keeping

Every jump in this chapter -- a Find hit, an outline click, a link -- adds to a browser-style **history** : **Alt+<-** goes back to the topic you came from, **Alt+->** forward again, across maps. It's the antidote to "where was I before I chased that link?"

And when a *view* itself is the asset -- zoomed to one corner, drilled into a branch, a filter applied -- save it: **View -> Save current view...** names the exact combination of pan, zoom, drill and filter, and re-applies it from the same menu in one click. A weekly review can open on precisely the view you triage from. Need to point someone else at a spot instead? **Copy link to this topic** puts a URL on the clipboard that opens the app on that map with that topic focused.

The strip at the bottom

A slim **status bar** sits bottom-centre of the canvas and answers the small questions before you ask them: how many topics are in view, how many are selected (click that count to zoom to the

selection), and the live zoom percentage (click it to snap back to 100%). At its left end sits a three-way **view switcher** -- **Map / Outline / Board** -- one click between the canvas and its two other faces, no menu required. And one quiet scale note belongs here: past roughly five hundred nodes the canvas starts rendering only what's near the viewport, so a big map stays fluid to pan and edit. You'll never see the seam -- it's simply why the three-hundred-node map doesn't wade.

A working rhythm for big maps

Put together, the loop looks like this: **collapse all** to see the shape, open the branch you need, **/** to jump to a specific node, work, **Fit** to re-orient, repeat. The outline panel stays open on the side as a table of contents. None of these moves is clever on its own; the habit of using them together is what keeps a three-hundred-node map feeling as manageable as a thirty-node one.

Your workspace, remembered

The panels you work with -- the **Outline** on the side, the **Notes** editor -- stay how you left them. Close the app with the outline open and it's open when you come back; the app remembers your panel layout between sessions, so you don't rebuild your workspace every morning. It's a small thing, but it's the difference between a tool that settles into your habits and one you have to re-arrange every time you sit down.

Now you try

Find (or build) a map big enough that it doesn't fit on screen. **Collapse all** , then open just one branch and talk yourself through it. Press **/** and jump to a node you only mentioned in a *note* -- prove to yourself that Find searches notes, not just topics. Open the **Outline** panel and use its filter to narrow a long map to a short list. Zoom right in on one node, then press **Fit** -- watch the whole map snap back into view; that's your "take me home". If you keep more than one map, open **All maps** and search for a term you know lives in a *different* one -- watch it open that map and land on the node. While you have two maps, add a **cross-map link** : point a node in one at the other, then click it and watch the app hop maps -- that's how a high-level map becomes a set of doorways into the detailed ones beneath it. While you're here, flip the **Layout** dropdown between the radial default, an org-chart, and a timeline -- same nodes, three readings -- and keep whichever makes the structure clearest. Then **reload the page** : the Outline panel is exactly where you left it, because the app remembered your workspace. The goal isn't to memorise the controls; it's to feel how much calmer a big map gets when you drive it at the right altitude instead of staring at the whole thing at once. Two more lenses: tag a handful of nodes `todo` / `doing` / `done` and hit **Board** to read the same map as columns; and, if you keep an overview map, bind a node to another via the **Roll-up** menu and click **Roll-ups** to pull its branches in -- a dashboard that refreshes itself. Save any filter you liked as a preset while you're at it.

A few more ways to cut a big map down to what matters. Open the **marker and tag index** and click a tag -- it lists every node carrying it and jumps you there, no scrolling. Turn on **auto-numbering** and the branches read 1, 1.2, 1.3 like a document outline, so you can talk someone through the map by reference. When a single branch is all that matters for the moment, **focus** it and the rest dims to quiet context. The **Power Filter** does the same by rule rather than by hand -- dimming (or,

with **Hide non-matches** ticked, removing from view) everything that lacks a marker, tag, or word, so the matches stand out while the map keeps its shape. Keep the **corner minimap** in view to hold your bearings while you are zoomed in. And put Find's forgiveness to the test on purpose: pick a long topic on your map and search for it *misspelled* -- drop a letter, swap two (`recieve` for `receive`) -- and watch the typo-tolerant pass still land you on the node. Once you've felt that, you'll stop composing careful queries and just type.

One more lens, the keyboard one. Press **Ctrl/Cmd + K** and first *do* something by name: type `collapse` and run **Collapse all branches** , then open the palette again and notice it now sits under **Recent** . Now *go* somewhere: type the text of a node you only mentioned in a *note* and watch **Go to:** land you on it, proving the palette searches notes too. Select a node and reopen the palette -- see the band of actions that act on just that node, and watch them vanish when nothing is selected. Then click the **?** button (or run **Keyboard shortcuts** from the palette) and skim the cheat-sheet once. If you have a phone handy, open the same map there and toggle the **Outline** panel -- it rises as a bottom sheet over a full-width canvas instead of squeezing the map.

Two altitude moves to feel before you go: run **Show level 1** and step the map open one level at a time, then **drill in** to a single branch, edit a node while you're inside, and press **Esc** -- the whole continent comes back with your edit in place. Ask a sharper question while you're at it: type `due:overdue tag:q3` into Find and watch a scan become a query. And when you've set up a view you'll want again -- a drill plus a filter -- **save it** from the View menu and re-apply it in one click.

This chapter grew a second toolbox; give it one pass too. In the **Outline** , double-click a row and rename it, then **Shift+Down** it past a sibling -- structure work without touching the canvas. Open **Map statistics** and read your map's word count and reading time; open **Relationships** and jump an arrow end to end. Filter to something real, tick **Hide non-matches** to feel the lights-off version, then **Extract matches to a new map** and watch a filtered view become its own document. Turn on the **Legend** and export a PNG that explains its own symbols. And on the way out, glance at the **status bar** : topic count, zoom, and the Map / Outline / Board switcher were under your cursor all chapter.

Part 3 turns outward: getting the map off your screen and in front of other people.

Sharing and exporting

Getting the map out -- losslessly when it matters

A map you can't get out of the app is a map held hostage. MindMap Studio is built on the opposite principle: your work is plain data, and there are many doors out. This chapter is the catalogue of them, and when to use which.

The export menu

The toolbar's **Export** menu offers, in rough order of fidelity:

- **JSON** -- the native format, and the only **lossless** one. Topics, notes, markers, images, boundaries, relationships, styling: everything in the model survives a round trip. Export to JSON for backup, for moving a map between machines, or any time you might want to re-open it later with nothing lost.
- **Markdown** -- the map as an indented outline (`#` root, nested bullets). Perfect for pasting into a document, a wiki, or a pull request. It's round-trippable as structure, though it naturally drops canvas-only detail like colours and arrows. (One thing it *keeps* when you ask: with outline numbering on -- Chapter 5 -- the numbers are baked into each line.)
- **OPML** -- the interchange format outliners speak. Use it to hand your structure to another outlining tool.
- **FreeMind / Freeplane (`.mm`)** -- the most widely-read mind-map format. Carries the topic tree, links, folded branches, and notes, so a map opens in FreeMind, Freeplane, XMind, and the many tools that import `.mm` . The bridge to almost any other mind-mapper.
- **Mermaid (`.mmd`)** -- the `mindmap` text format you embed in Markdown, a README, or a wiki that renders Mermaid (GitHub, GitLab, Notion, many docs tools). For when the map should live *as text* inside something you're already writing.
- **XMind (`.xmind`)** -- the native format of one of the most popular mind-mappers, written the modern (2020+) way. Carries the topic tree, notes, links, and tags, plus floating topics and relationships, so the map opens natively in XMind rather than going through `.mm` .
- **SimpleMind (`.smmx`)** -- the native format of the cross-platform SimpleMind app. Carries the topic tree, notes, web links, and relations, plus floating topics, so the map opens natively in SimpleMind on desktop or mobile.
- **MindManager (`.mmap`)** -- MindManager's own format. Writes the topic tree, notes, hyperlinks, stock icons, relationships, and the two-sided left/right arrangement -- the mirror of the `.mmap` importer, so a map you started here can go back to a MindManager user.
- **PNG** and **SVG** -- the map as a picture. PNG for slides and chat -- in plain, sharp **@2x** , print-grade **@4x** , and **transparent** variants -- and SVG when you want a crisp, scalable image that survives zooming. There's also **Copy image to clipboard** , which skips the file entirely and pastes straight into a chat or a deck.
- **HTML** -- a self-contained web page of the map, openable in any browser with nothing installed.
- **Interactive HTML** -- the same one-file, opens-anywhere idea, but *navigable* instead of a picture: the map becomes a collapsible, searchable outline. The recipient folds branches by clicking a topic, expands or collapses the whole thing at once, and types in a filter box to

highlight and narrow to matching topics (plus Ctrl/-scroll to zoom and drag to pan). It's all one file with the data, styling, and a tiny script inlined -- no app, no server, no internet -- so it's the format for handing a *big* map to someone who needs to explore it, not just look at it.

- **HTML slide deck** -- the walk-through (Chapter 7) as a standalone, navigable presentation in a single file: an overview slide, then one per branch -- each drawn as its **actual map image** , not a bullet list -- advanced with the arrow keys or a click, speaker notes a keypress away. Hand someone the file and they can present your map with nothing installed.
- **PowerPoint (`.pptx`)** -- a real, editable slide deck: an overview slide, then one per branch rendered as its **live map image** , with each topic's note carried into the slide's speaker notes. For when the deck has to live in PowerPoint -- a house template to apply, or a room to run from the corporate machine.
- **Word (`.docx`)** -- the map as an editable outline document: a title, indented bulleted topics, and notes as italic lines. For when the next step lives in a word processor -- minutes, a brief, a hand-off to someone who doesn't use the app.
- **Excel (`.xlsx`)** -- the map as an indented outline worksheet: each topic in the column matching its depth, with a Notes column. For when you want to sort, filter, or count the outline as a spreadsheet.
- **PDF** -- a real `.pdf` file, written directly (sized to the map, or **A4** / **Letter**), or the classic route through the browser's print dialog when you want its options.

A rule of thumb: **JSON to keep it, Markdown to discuss it, PNG/SVG/HTML to show it, interactive HTML to let someone explore it, the slide deck or PowerPoint to present it, Word or PDF to send it, Excel to crunch it.**

And the scope is yours: **right-click any topic -> Export this branch...** exports just that subtree, in any of the picture/document formats, framed to its own bounds -- for the one branch that needs to travel without the map around it.

A worked tour of the interchange exports, one file each. Take a project map and pick **Export -> OPML** ; open the `.opml` in your outliner and the topics arrive as nested headings, ready to keep editing as an outline. **Export -> FreeMind (`.mm`)** and double-click the file: it opens in FreeMind, Freeplane, or XMind with the tree, the folded branches, the links, and the notes intact -- the one bridge almost every mind-mapper reads. **Export -> Mermaid (`.mmd`)** gives you a `mindmap` block you paste straight into a README on GitHub, where it renders as a diagram in the rendered Markdown. **Export -> XMind (`.xmind`)** opens natively in XMind 2020+ -- tree, notes, links, tags, floating topics, and relationships -- with no `.mm` detour. **Export -> SimpleMind (`.smmx`)** opens natively in SimpleMind on desktop or phone, carrying notes, web links, and relations. And **Export -> HTML** hands someone a single self-contained page that opens in any browser with nothing installed -- one file, no app, no server. Same map, six more doors, and every one is a file on *your* disk.

Copy the outline

Sometimes you don't want a file at all -- you want the text on your clipboard, ready to paste. The **Copy outline** button copies the whole map as a Markdown outline in one click: drop it straight into an email, a chat message, a ticket, or a document. It's the same structure as the Markdown export

without the round trip through a saved file -- the fastest way to turn a map into words somewhere else. Reach for it when the map was the *thinking* and some other place is where the writing has to land.

Its sibling **Copy as table (TSV)** (in the **More** menu) flattens the tree into rows instead: one per topic, with **Topic / Depth / Note / Tags** columns, tab-separated so it pastes straight into a spreadsheet *as columns* . It's the bridge from map to grid -- tag topics with owners while you plan, copy as a table, paste into the sheet, and the per-owner breakdown is a pivot away. (Chapter 8's decision recipe reaches for this map-to-grid bridge when a comparison outgrows the map.)

Importing

The same breadth applies going in. MindMap Studio reads:

- **Native** `.json` -- the lossless round trip of the export above.
- **Markdown** outlines -- any `#` /bullet structure becomes a map, **Markmap** -flavoured Markdown (YAML frontmatter plus multi-level headings) included.
- **OPML** -- from other outliners.
- **FreeMind / Freeplane** `.mm` -- topics, links, folded state, and notes.
- **Mermaid** `mindmap` text -- any node shape; the hierarchy comes from the indentation.
- **XMind** `.xmind` (2020+) -- topics, notes, web links, and labels become tags.
- **SimpleMind** `.smmx` -- topic tree, notes, web links, and relations.
- **MindMup** `.mup` -- the JSON maps from the browser-based MindMup.
- **TextBundle / TextPack** (`.textpack`) -- the bundle's Markdown becomes the map; what Bear, Ulysses, and iA Writer export.
- **MindManager** `.mmap` -- see below.

You can **batch-import** several files at once, which turns a folder of outlines into a library of maps in one step. (The list keeps growing -- Word `.docx` , Excel `.xlsx` , iThoughts `.itmz` , MindMeister `.mind` , and older XMind files all open too.) These bridges cover the common ground between tools; each keeps the topic tree and the fields that map cleanly, and -- like the `.mmap` importer -- quietly leaves behind only the tool-specific extras it can't represent.

The **Excel** door deserves a sentence of its own, because it's how a *spreadsheet-shaped* plan becomes a map: the importer reads the first sheet as an indented outline -- the column of a row's first filled cell is its depth (column A a root, B its child, and so on), and a later cell on the same row comes along as that topic's note. It's exactly the shape the Excel *export* writes, so the worksheet a colleague reorganised can come back in as the map it started from.

Not everything you want to map arrives as a file, though. Often it's just *text* -- an agenda in an email, a list in a chat, the bones of an outline you typed somewhere else. **Paste text** takes that straight from the clipboard: paste it in and the indentation (or `#` heading levels) becomes the tree, bullet and number markers are stripped, and you get topics. It reads Markdown's shorthand on the way through: a `[ ]` or `[x]` checkbox becomes a real task (to-do or done), a `[text](url)` line becomes a topic with the link attached, and stray `**bold**` markers are cleaned off rather than pasted in. A rectangular *spreadsheet* selection pastes just as well -- rows become topics, with the

extra columns carried along. Drop it in as a new map, or **Add under selected** to graft it onto a branch you're already growing. It's the lowest- friction on-ramp there is -- and, because it never leaves the browser, the private way to bring in an outline you drafted anywhere, including one a chatbot wrote for you.

The MindManager bridge

If you're arriving from MindManager, the `.mmap` **importer** is the on-ramp. It reads a `.mmap` file and recovers the parts that map cleanly onto MindMap Studio's model: the topic tree and text, notes, stock icons (rendered as emoji markers), hyperlinks, relationships, boundaries, and floating topics. It was built against MindManager's own published schema rather than guessed at, so the common cases come through faithfully.

There is now also an **off-ramp** : **Export -> MindManager (`.mmap`)** writes the same parts back out -- tree, notes, hyperlinks, icons, tags, task info (dates, priority, progress), embedded images, relationships, and the two-sided left/right arrangement -- so a map you built here can land on a MindManager user's desk. It is the mirror of the importer, and a map round-trips back into MindMap Studio with those fields intact. MindManager is strict about its format, so confirm an exported file opens in your MindManager version before relying on it.

Both directions are deliberately **lossy on project data** : MindManager carries task scheduling, resources, and Gantt information that MindMap Studio doesn't model, and the importer tells you when it has left something behind rather than pretending the map came across whole.

> **Why lossy is the honest choice.** A converter that silently drops what it can't represent leaves you to discover the gaps later, usually at the worst moment. The importer's warnings are a feature: they tell you exactly what to check.

Going back in time

Exporting a `.json` is a snapshot you take on purpose; **version history** takes them for you. The **History** panel keeps a running list of past states of the current map -- captured quietly as you edit (every few minutes, not every keystroke) and whenever you click **Save version now** . Each entry shows when it was taken and how big the map was; **Restore** rolls the map back to it. The safety net under the safety net: restoring first checkpoints what you have *now* , so even an unwanted restore is one click from undone. It's all local (capped at the 30 most recent, kept in the browser's database) and travels with nothing -- a private undo that outlives the session, where the in-session Undo (Chapter 1) stops at the last reload. Think of Backup as the whole-library snapshot and History as the per-map flight recorder.

There's also a **Play timeline** button: instead of eyeballing the list, play the snapshots back in order and *watch* the map grow on the canvas -- step frame by frame, drag the scrubber to any point, or let it auto-play. It's read-only while you watch, so nothing changes until you pick a frame and hit **Restore this** (or **Exit** , or Esc, to drop back to the live map). On a map you've been building for weeks, it's a quietly satisfying way to see how the thinking took shape.

A map as a file on disk

Between "lives in the browser" and "exported once" there's a third way to keep a map: link it to a **file on disk** . MindMap Studio's native file is `.mmst` (the lossless JSON format under a distinct extension). **Open file...** (Ctrl+O) opens one; **Save to file** (Ctrl+S) and **Save as...** write one; an **Open recent** list brings back the files you use often. Once a map is linked to its file, the same continuous autosave **writes through to disk** as you edit -- so a map kept in a synced folder (Dropbox, OneDrive, a git repo) stays current without a Save ritual. The app is careful at the edges: if the file changes on disk underneath you -- edited elsewhere, or synced in -- background saving pauses and an explicit Save asks before overwriting, and if the same map is open in two tabs, you're warned before the autosaves can fight. (Save-in-place needs a Chromium desktop browser; elsewhere, Save downloads the file and your work still autosaves to the browser.)

Where your data lives

Nothing in this chapter sends your map anywhere. Exports are files saved to *your* machine; imports read files *you* choose. Day to day, every map lives in your browser's local database, and the app works fully offline (Chapter 7). Sharing is always an explicit act, never a background one -- which is exactly how a thinking tool should treat the half-formed ideas you trust it with.

Because the app is a PWA, that locality holds even with no network. Load it once and it keeps working offline: the app shell is precached, so on your next visit -- plane, train, dead Wi-Fi -- it opens and your maps are right there, with a quiet **Ready to use offline.** note the first time the cache lands. When a new version ships, it never reloads under you mid-edit; instead a small **A new version is available.** toast appears with a **Refresh now** button, and the swap happens only when you click it. And on a phone the editor adapts -- the wide toolbar collapses into a single compact, horizontally-scrollable strip and the side panels slide up as bottom sheets -- so the same map is workable on a screen it was never drawn for.

Now you try

Take any map and export it twice: once to **JSON** and once to **Markdown** . Open the Markdown in a text editor — there's your outline, ready to paste into a document. Now start a **new** map and import the JSON you saved: it comes back exactly, down to the markers and arrows. You've just proved the thing that matters most about a thinking tool — your work isn't trapped in it. **JSON to keep it, Markdown to share it.** Make that round trip once and you'll trust the app with the ideas you're still figuring out.

Then take the same map to an audience two more ways. Export the **slide deck** (or **PowerPoint** , if the deck has to live there) and open it -- you're presenting: an overview, then one slide per branch, arrow keys to move, nothing installed. Export **Word** and open it in your word processor -- the same map as an editable outline, ready to become minutes or a brief. One map, four jobs -- archived, discussed, presented, and written up -- and not once did your work leave your machine.

Now get it out as a *picture* and as a *paste* . Export **PNG** (or **SVG** for a crisp, scalable version) and drop it into a slide or a chat -- notice the labels, icons, arrows and boundaries all render, because the image carries real text, not a screenshot. Export to **PDF** when it needs to print, or to **Excel**

when you want to sort and count the outline as a spreadsheet. For a big map someone needs to *explore* rather than glance at, export **Interactive HTML** and open it: same single offline file, but now you can fold branches and type in the filter box to narrow to what matters -- proof the export can carry the *navigation* , not just the picture. Finally, when you just need the words somewhere *now* , click **Copy outline** and paste -- no file, no download, just the map as a Markdown outline wherever your cursor is; or **Copy as table (TSV)** and paste into a blank sheet to watch Topic / Depth / Note / Tags land as real columns. Same map, every door out: a file to keep, a picture to show, a sheet to crunch, a paste to drop.

One round trip is worth running deliberately: export your map to **Excel** , move a row or edit a note in the spreadsheet, and **import** the `.xlsx` back -- the column-indented sheet returns as the tree it described. That's the full map-to-grid-to-map loop, and once you've made it, "can you put that in a spreadsheet?" stops being a rewrite.

One safety net is worth feeling before you trust a map with real work: make a few edits, then open **version history** and restore an earlier snapshot -- the map rewinds intact, and you can **play the timeline back** to watch it grow from the first node to now. And when you bring a map in from MindManager, notice that its stock icons arrive as familiar **emoji** -- the meaning crosses over even when the file format doesn't.

Now prove the *interchange* . Export the map to **OPML** and reopen the `.opml` in your outliner -- the headings are your topics. Export **FreeMind (`.mm`)** and open it in Freeplane; export **Mermaid (`.mmd`)** and paste the `mindmap` block into a README so it renders as a diagram; export **XMind** or **SimpleMind** and watch it open natively in those apps, notes and links along for the ride; export self-contained **HTML** and double-click it in any browser. Then go the other way: take that `.mm` (or a `.opml` , `.xmind` , `.smmx` , `.mup` , `.itmz` , `.mind` , `.xlsx` , `.docx` , `.textpack` , a MindManager `.mmap` , a plain or Markmap-flavoured `.md` , or a Mermaid file) and **Import** it back -- the tree and the fields that map cleanly come across -- or drop a whole folder at once to **batch-import** them into a library. When you've only got loose text -- an agenda from an email -- **Paste text** turns the indentation into a tree in one click. Finally, feel the PWA: kill your network and reload (it still opens, maps and all, because the shell is precached), watch for the **A new version is available.** toast next time a build ships, and open the same map on your phone -- the toolbar shrinks to one scrollable strip and the panels become bottom sheets.

The final chapter is about the most demanding kind of sharing: standing up and walking a room through a map live.

Presenting and workshops

The library, templates, walk-through mode, and running a room

A map is often built to be shown. This chapter covers the features that turn a private canvas into a shared session: managing many maps, starting from the right template, presenting branch by branch, and the offline reliability that lets you do it anywhere.

The library: many maps, one app

MindMap Studio keeps a **library** of named maps in your browser. The map switcher in the toolbar moves between them; **+ New...** starts another; you can **duplicate** the current map (to try a variation without risking the original) and delete ones you're done with. Because each map is independent and locally stored, the library is your filing cabinet -- one map per project, per meeting, per idea -- with cross-map links (Chapter 5) tying related ones together.

As the cabinet fills, the Start screen's **All maps** view keeps it navigable: a **search box** filters by title, **pinned** maps float to the top of every list (star the handful you live in), and named **folders** group the rest -- a map lives in one folder or none, and deleting a folder releases its maps rather than taking them down with it. Deleting a *map* is just as forgiving: it's a soft delete into the **Trash** , where it waits until you restore it or empty the trash for good. Two commands keep the maps themselves well-factored: **New map from this topic** promotes an overgrown branch into its own map, and **Insert map as branch** grafts a small map back into a bigger one.

Templates: a running start

A blank canvas is the right start for a freeform brainstorm. For everything else, **+ New...** offers templates:

- **Blank** -- a single root, for when the structure is entirely yours.
- **Brainstorm** -- the central idea with the six question-branches from Chapter 2.
- **SWOT** -- Strengths, Weaknesses, Opportunities, Threats, ready to fill in.
- **Project plan** -- Goals, Scope, Milestones, Risks, Team.
- **5 Whys** -- a nested root-cause chain; **Decision** -- pros and cons side by side; **Retrospective** -- Start / Stop / Continue; **Meeting notes** ; and **Pre-mortem** -- "it failed: why?"

A template is just a starting structure you then make your own; its value is saving you the first thirty seconds of "what were the boxes again?" and nudging you toward a complete frame. (Mid-map, the same frames are available as **map parts** -- **Insert -> Map parts** drops a SWOT, a pros-and-cons pair, a 5W1H, or a meeting agenda *under the selected topic* , so a framework can be a branch, not only a map.)

Where a template is an *empty* frame, the **Examples** group in the same **+ New…** menu gives you twenty *complete* , worked maps -- a filled launch plan, a sprint retro, a worked SWOT, a flowchart, a whiteboard, a trip itinerary, a study map, a **GTD Areas of Focus** overview (Chapter 8 builds its atlas pattern on it) -- to open and adapt rather than build from scratch. They're the fastest way to

see what a finished map looks like, and to learn a feature by reading one that uses it.

Time-boxing a brainstorm

Ideas come faster under a gentle clock. The **brainstorm timer** in the toolbar is a small time-box: pick a length -- a few minutes -- start it, and generate hard until it rings. A divergent sprint ("every idea, no judging, go") works far better with a visible countdown than an open-ended "let's brainstorm", and a workshop gets a shared rhythm from it: sprint to capture, stop to cluster and prune, repeat. It's a nudge rather than a rule -- but the nudge is most of why time-boxing works.

Walk-through presentation mode

When it's time to present, **Present** enters a focused **walk-through** mode. Instead of showing the whole map at once and asking the room to find the thread, walk-through moves through the map branch by branch, framing each in turn, so attention follows you. Combined with a dark theme (Chapter 4) and a collapsed starting view (Chapter 5), it turns a dense canvas into a guided tour: open the branch under discussion, talk to it, move on.

Presenting from the map itself -- rather than a deck exported from it -- keeps the single source of truth live. A question sends you to a different branch; an idea from the room becomes a node on the spot. The map is the slides *and* the working document.

Two stagecraft keys are worth knowing before the lights come up: **B** drops a black curtain over everything and **W** a white one -- the classic "eyes on me, not the screen" move -- and any key or click lifts it.

The guided walk: touring the live canvas

Present turns the map into slides; the **guided walk** keeps you *on the canvas* . Start it (**Guided walk** in the menu, or from the command palette) and a small bar appears: each step spotlights one topic in reading order, dimming the rest, with that topic's **note** shown as your talking points and the arrow keys stepping forward and back. Because it's the live canvas, you can stop mid-walk, edit the node under discussion, and walk on. A **cinematic** toggle on the bar swaps the flat step for an animated zoom that frames each branch as it arrives -- the difference between a laser pointer and a camera move; the app remembers which you prefer. Reach for the walk in working sessions, where the map must stay editable; reach for Present when the room expects slides.

Presenter view: what only you see

A slide is for the room. But you, the presenter, usually want more in front of you: the point you meant to make, where you are in the running order, and what's coming next. Press **P** (or click **Presenter view** in the control bar) and a sidebar opens beside the slide with exactly that -- visible to you, invisible to the audience, who still see only the slide.

Four things live in that sidebar. Your **speaker notes** come first: whatever you wrote in the current branch's note (Chapter 4) shows here, formatted, so your talking points travel with the map instead of on a separate sheet -- and if a slide has no note, it simply says so. Below that, **Next up** names

the slide you're about to advance to, so you can set up the transition before you make it -- or see "End of map" and know to land the close. A **Timer** keeps you honest: an elapsed clock runs in the footer for the room, and here you can give the talk a **budget** in five-minute steps -- the clock stays green while you're comfortably inside it, turns amber in the final stretch, and red once you're over, with a "-2:30 left" readout that makes pacing a glance rather than arithmetic. Last is the **Agenda** : the map of your whole talk, every slide in order with the current one lit and a "3 / 8" marker for your place in it. It's not just a readout -- click any line to jump straight there, which is how you handle the question that belongs three branches away and the "can you go back to that one?" without losing your footing.

It all sits on one screen -- there's no second window to wrangle -- and toggling it changes nothing for the room. Press **P** again to hide it. The habit worth forming: before you present, drop a sentence or two into the note on each branch you'll speak to. When the lights are on you, your script is already there.

It works offline, and it installs

MindMap Studio is a **progressive web app** . The first time you load it, it caches its own shell, so afterwards it opens and runs with **no network at all** -- on a plane, in a basement meeting room, anywhere. You can also **install** it to your desktop or home screen, where it launches in its own window like a native app. For a tool you might open to capture a thought the instant you have it, "always available, never loading" matters.

Because it caches itself, it also **updates itself politely** : when a new version ships, the running app shows a small "new version available -- Refresh now" prompt rather than reloading under you mid-thought. Click it when you're ready; an in-flight edit is never lost.

On the device in your hand

Because it's a web app, it goes where you do. On a **phone or tablet** the layout adapts -- the toolbar compacts to a scrollable strip and the side panels rise as a **bottom sheet** instead of squeezing the canvas -- so you can capture an idea or pull up a map on the move and open it on a laptop later. Same app, same maps, sized for the screen you're holding; paired with the offline caching above, it makes "the map is wherever I am" simply true.

A room you can't choose

Present often enough and you'll meet the projector that washes colour out, the huge screen where every animated pan swings like a camera crane, and the colleague for whom motion is genuinely unpleasant rather than slick. Two settings (**Ctrl+,** , under Appearance) exist for exactly this. **Reduce motion** makes zoom, fit, and the guided walk *instant* instead of animated -- set it to **On** for the big screen or a motion-sensitive audience, or leave it on **System** and it follows each viewer's own operating-system preference. **High contrast** strengthens borders, dividers, text and focus rings the same way -- **On** for the washed-out projector, **System** to respect the reader's setting.

The canvas also carries its structure where sight doesn't reach: the overlays that exist only as

drawings for sighted readers -- relationships, boundaries, summaries, callouts -- are exposed to **screen readers** as navigable lists ("from Budget to Risk", with the label), so the map's cross-links survive the trip into assistive tech. None of this needs to matter until the day it's the only thing that does; it's the difference between a map you can present and a map anyone can be presented to.

Back up the whole library

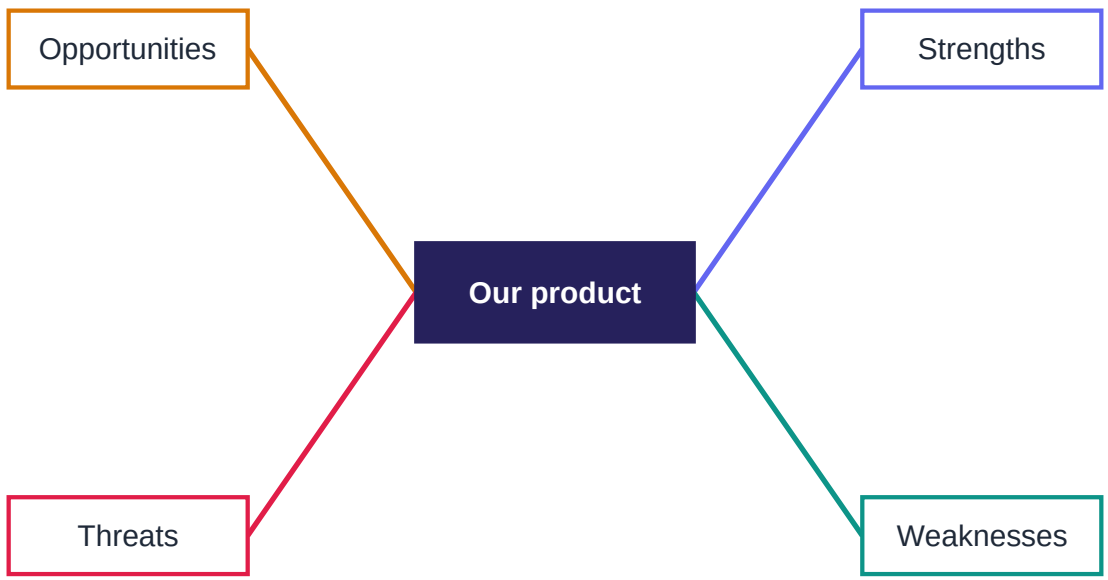
Individual maps export as JSON (Chapter 6). The whole **library** can be backed up and restored in one operation -- a single file with every map in it. Run a backup before a big reorganisation, before switching machines, or just on a schedule you trust. It's the belt to local storage's braces: your maps live on your machine, and a backup means a machine is not a single point of failure.

Running a session: a short playbook

1. **Before:** build or open the map; **collapse all** so it opens at altitude; pick a theme that suits the room.
2. **Open: Fit** , then enter **Present** .
3. **During:** walk branch by branch; capture new ideas as nodes as they come; use / to jump when the conversation does.
4. **After:** export to **PNG** or **PDF** for the recap; export **JSON** or run a **library backup** to keep the working map safe.

Now you try

Start a new map from the **SWOT** template. It opens with four branches ready to fill:



A SWOT map: a subject in the centre, four branches to fill in — the SWOT template.

Pick something real -- a product, a project, a decision -- put it in the centre, and spend ten minutes

filling each branch. (Start the **brainstorm timer** for those ten minutes -- a visible countdown makes the sprint sharper than an open-ended "let's fill it in".) Don't aim for a long list; aim for the *honest* three items per branch. Then look across the four: does an Opportunity answer a Weakness? Does a Threat undercut a Strength? Those cross-links are where a SWOT stops being four lists and starts being analysis -- draw a **relationship** arrow for each one you find. That move, more than the lists themselves, is the thinking.

Then rehearse the room. **Duplicate** the map so you can experiment freely, switch back to the original from the map switcher, and hit **Present** : walk-through frames each branch in turn -- arrow keys to move, the room's attention following yours, one branch at a time. Press **P** to flip on the **presenter view** -- your speaker notes, the next slide, and the agenda on your side of the screen only, the room still seeing just the slide. When you're done, run a **library backup** -- one file holding every map, the belt to local storage's braces -- and, if you haven't already, **install** the app to your desktop so it's one click away and runs with no network.

Before you leave the stage, try the other one. Start a **guided walk** and step the spotlight through your SWOT with the arrow keys -- edit a node mid-walk to feel that the map stays live -- then flip the **cinematic** toggle and watch the step become a camera move. Back in **Present** , open presenter view and give yourself a ten-minute **budget** : the clock runs green, amber, red as you talk, and **B** blanks the screen when you want the room's eyes on you.

And once, before a real session, open **Settings** and flip **Reduce motion** to On, then step three slides -- the spotlight now *cuts* instead of glides. Some rooms want the glide; some want the cut; know which switch gives you each.

Prefer a running start over a blank SWOT? Open an **Example** instead (**+ New… -> Examples**) -- a filled launch plan, a retro, a trip, a worked SWOT -- and adapt it. Same skills, less blank page; it's also the quickest way to see a feature used in anger.

That's the whole tool, end to end -- from a single node in Chapter 1 to a map you can think with, enrich, navigate, share and present. One chapter remains, and it's about none of the buttons: Chapter 8 takes the method itself and applies it, job by job, to the work you'll actually bring to a map.

Thinking with maps

The method applied: six jobs, one technique

The first seven chapters taught the parts and the tool. This one is about the *practice* -- the recurring jobs you'll actually bring to a map, and the shape of a good session for each. None of it introduces a new feature; every move below was taught somewhere in Chapters 1-7, and is referenced back to where. What's new is the choreography.

Each job follows the same deep pattern -- **capture, structure, enrich, use** -- but the proportions differ wildly. A brainstorm is nearly all capture; a decision is nearly all structure; a plan lives in the enrichment. Knowing which phase a job leans on is most of knowing how to run it.

The brainstorm: quantity first, honestly

The shape: the subject in the centre, and around it either nothing (a true open brainstorm) or the six question-branches of the **Brainstorm** template (Chapter 2) when you want gentle rails.

The session. Set the **brainstorm timer** (Chapter 7) for ten minutes and stay in the divergent gear the whole time: **Quick add** (Chapter 1) -- type, Enter, type, Enter -- and every idea lands as a node, judgement suspended. Misparented, misspelled, half-formed is all fine; the only failure is an idea that got away. If a thought arrives that belongs to some *other* subject entirely, drop it as a **floating topic** (Chapter 1) off to the side and keep moving.

When the timer rings, switch gears and *say so* -- even alone, the announcement matters. Now: **drag** related ideas together (Chapter 3), merge duplicates, promote the labels that name a cluster, and prune the noise. Expect the map to earn two or three relationship arrows and at most one boundary. Finish by asking the emptiest branch one question out loud -- the gap is usually more informative than the crowd.

Done looks like: ten to thirty keyword nodes in three to six named clusters, and one insight you didn't have before -- typically a connection between two clusters, which is why the method insists both end up on the same page.

Notes on a book: the map as a memory

The shape: the book (or course, or talk) in the centre; one branch per chapter or per *question you brought to it* -- the second is better. `What does she claim?` , `What's the evidence?` , `What do I disagree with?` pull more from a text than chapter headings do (rule 3, Chapter 2).

The session. Read with the map open and capture in **keywords only** -- the one-to-three-word discipline (rule 1) is at its most valuable here, because compressing an author's paragraph to two words *is* the act of understanding it. Quote-worthy passages go in a **note** behind the node (Chapter 4); the label is your compression, the note is the source. Mark the ideas you don't yet understand with a **question-mark marker** (Chapter 4) so your confusion is queryable later -- before a second reading, filter to the question marks (Chapter 5) and read only for them.

The recall trick. A map you made is a memory you can test. Later -- a day, a week -- **collapse all** (Chapter 5), then, *before* expanding each branch, say out loud what's under it. Expand and check. The branches you couldn't reproduce are the ones to re-read; the spatial layout means you'll often remember *where* an idea was before you remember what it said, and the where drags the what back with it.

Done looks like: a one-screen map you can re-explain the book from a month later -- which is a different artifact from highlights you'll never reopen.

The meeting: capture in front of the room

The shape: the **Meeting notes** template (Chapter 7), or an agenda you build the night before -- one branch per agenda item, in speaking order (the **right-only layout** , Chapter 3, reads well for this).

The session. Share your screen and take the notes *as the map, in front of everyone* . This is the honest version of minutes: when a decision lands as a node while the room watches, everyone has agreed what was decided -- there's no write-up to dispute on Thursday. Tag action items with their owner (`#anna` , `#you` , Chapter 4) and add a **due date** where one was actually agreed (Chapter 4). New topics that hijack the agenda get captured as nodes under a `Parking lot` branch and *visibly parked* -- the map lets you honour an idea without following it.

Afterwards takes two minutes, because the notes already exist: filter by an owner's tag to read someone their actions (Chapter 5), then **Copy outline** (Chapter 6) into the follow-up email, or export **Word** if the minutes must be a document. The map remains the source of truth; next week's meeting starts by reopening it and walking the action items.

Done looks like: the meeting ends and the minutes are *already sent* .

The decision: structure until the answer is visible

The shape: the decision in the centre, stated as a question -- `Which vendor?` not `Vendors` . First-level branches: the **options** , plus one branch named `Criteria` -- what actually matters, agreed *before* the arguing starts.

The session. This job is nearly all convergent gear. Under each option, capture the honest case -- for, against, unknowns -- in keywords. Then make the structure do the analysis: give the deciding criteria a **marker** each and pin them onto the options that satisfy them (Chapter 4); draw **relationship arrows** for the tensions -- the cheap option that *blocks* the integration you need, the fast one that *depends on* a hire you haven't made (Chapter 3, typed relationships). Cap each option's branch with a **summary bracket** stating its verdict in five words (Chapter 3).

Then apply the method's honesty rule: **when the comparison gets numerical, leave the map** (Chapter 2, "when not to map"). If you're weighting six criteria across four options, export the structure to **Excel** (Chapter 6) and score it as a table. The map found the options, the criteria, and the tensions -- that was its job; the arithmetic belongs in a grid.

Done looks like: a map where the answer is *visible* -- one option's branch carrying the markers,

the fewest red arrows, and a summary you believe -- or a clean handoff to a spreadsheet, with nothing important missing from its rows.

The plan: a map that tracks itself

The shape: the **Project plan** template (Chapter 7) -- Goals, Scope, Milestones, Risks, Team -- or the conference map you've grown since Chapter 1.

The session. Plans are where a map earns its keep over weeks, so the enrichment phase dominates. Make work explicit: **due dates** , **priority** , and **progress** on the topics that are actually tasks (Chapter 4), and let the roll-up report each branch's health upward. Encode the plan's risk vocabulary once as **conditional rules** -- "anything tagged `risk` goes red", "anything overdue turns amber" (Chapter 4) -- so the map re-paints itself as reality changes. When the schedule slips a week, shift the whole branch's dates in one move rather than editing them singly (Chapter 4).

Then *drive* it with the lenses from Chapter 5: `due:overdue` in Find is your Monday morning; the **Board** grouped by schedule is your standup; a **saved view** -- the filter and zoom you triage from -- makes the weekly review one click. Presenting status upward is Chapter 7's walk-through over the same living map: collapse to milestones, open the branch under discussion.

Done looks like: a plan you *consult* instead of maintain -- the difference between a map with task fields and a status document is that the map answers "what's late?" without being re-written.

The root cause: asking why in a tree

The shape: the problem in the centre, stated as the *symptom* -- `Deploys keep failing` -- and the **5 Whys** template (Chapter 7) or the **fishbone layout** (Chapter 5) as the frame. The fishbone suits a problem with parallel contributing categories (people, process, tooling, environment); the Whys chain suits a single stubborn thread.

The session. Each answer to "why?" becomes a child; each child gets asked again. The tree keeps you honest in a way linear notes can't: a *genuine* root cause sits at the end of a chain, but the map also shows you when two chains **converge on the same cause** -- draw the relationship arrow, because a cause that explains two symptom branches is where the fix pays twice. Mark dead ends with a cross rather than deleting them; in a post-mortem, the exonerated suspects are part of the record.

Done looks like: a chain you can read backwards -- "fix *this* , and that, then that, stops happening" -- and at most two nodes marked as the actual work to do.

The atlas: a library that thinks together

One map per subject is the rule (Chapter 7's library), but subjects connect -- and the method scales past a single canvas.

The shape: many working maps, plus one deliberately thin **overview map** whose first-level branches are your areas of responsibility. The worked **GTD Areas of Focus** example in the gallery (Chapter 7) is exactly this pattern, filled in: open it, read it, then build your own.

The practice. Nodes in the overview are **doorways** : give each a cross-map link to its detailed map (Chapter 5), and let **roll-ups** (Chapter 5) pull the live top level of the busiest maps into the overview so the dashboard stays current without copying. Stray thoughts go to the capture **inbox** (Chapter 1) the moment they occur, unsorted. Then keep one small ritual: a **weekly sweep** , where you empty the inbox onto the maps where each item belongs, walk the overview's branches, and ask each area the same question -- "what here is stale?" The created/modified timestamps (Chapter 4) answer better than memory does. Ten minutes a week is enough, and it's the difference between a library and a landfill.

Done looks like: any thought you've had in the last month is findable in under ten seconds -- **All maps** (Chapter 5) for content, the overview for structure.

The common thread

Six jobs, one loop. *Capture* without judging, at whatever speed the ideas arrive. *Structure* as a separate act, where the conclusions form. *Enrich* only what carries meaning -- a marker vocabulary, dates on real tasks, notes behind short labels. *Use* the map through lenses -- filter, focus, board, walk-through -- rather than by staring at all of it. The features change from job to job; the loop doesn't.

And one meta-rule spans them all: **the map is a means.** The brainstorm exists for the insight, the study map for the recall, the decision map for the choice. When a map has done its job, export what the next step needs (Chapter 6), and let the map retire with the version history as its memoir. A tool for thinking earns its place by being easy to leave.

Now you try

Pick the job on this list you actually have this week -- not the one that sounds most interesting -- and run its recipe end to end, timer and all. One honest run through one recipe will teach you more than re-reading the chapter five times.

Then, whichever job you picked, close the loop the same way: when the session ends, ask the map three questions. *What did I learn that I didn't bring?* (If nothing: you skipped the convergent pass -- go back and structure.) *What's the smallest version of this map that still carries it?* (Prune to that.) *What does the next step need?* (Export exactly that, and nothing more.)

That's the whole method. The appendices that follow are reference; the map you open tomorrow is the actual last chapter.

Appendix A -- Keyboard reference

The fastest way to build a map is to keep your hands on the keys. This is the full set, grouped by what you're doing. (Ctrl reads as Cmd on a Mac throughout. The same list lives in the app -- the **?** button, or run "Keyboard shortcuts" from the command palette -- and is generated from the bindings themselves, so it can't drift.)

Building structure

- **Enter** -- add a sibling (a node at the same level as the selected one).
- **Tab** -- add a child (a node one level deeper); **Ctrl+Enter** does the same.
- **Shift+Tab** -- outdent: promote the node one level back toward the root.
- **Delete** (or Backspace) -- remove the selected node and everything beneath it.
- **Ctrl+Shift+Up / Down** -- move the node up / down among its siblings.
- **Alt+Shift+Left / Right** -- promote / demote the node.
- **Ctrl+C** -- copy the selected branch (or every selected branch); **Ctrl+Shift+V** -- paste them under the selection (the branch clipboard survives switching maps); **Ctrl+D** -- duplicate the branch as a sibling.

Editing

- **Double-click** a node, press **F2** , or just start typing on a selected node to edit it.
- **Ctrl+B / I / U** (while editing) -- bold / italic / underline the selection.
- While editing: **/** at the start opens the command menu, # picks a tag, **[[** or @ links to a topic or map by name.
- **Ctrl+V** (node selected, not typing) -- smart paste: an image, an outline, a URL, or a spreadsheet selection, routed automatically.
- **Ctrl+Shift+1..9** -- set the node's task priority (1 = highest).
- **Ctrl+Z** -- undo (model-aware: it reverses the real edit); **Ctrl+Y** or **Ctrl+Shift+Z** -- redo.

Finding and moving around

- **/** or **Ctrl+F** -- open Find and Replace (searches every field a topic carries; operators like `tag:` , `priority:` , `due:overdue` , `has:note` , `-exclude` sharpen it).
- **Enter / Shift+Enter** (in Find) -- next / previous match.
- **Arrow keys** -- walk the selection: Left = parent, Right = child, Up/Down = siblings.
- **Alt+Left / Alt+Right** -- back / forward through the topics you've visited.
- **Ctrl+K** -- the command palette: run any action, jump to any topic, switch to any map.
- **Ctrl+Shift+L** -- start a relationship from the selected node (Enter completes it).
- **Drag a node** onto another to re-parent it; **Shift-drag to empty canvas** detaches it; **Ctrl-drag onto a node** drops a copy.

Viewing

- **Ctrl+Plus / Ctrl+Minus / Ctrl+0** -- zoom in / out / reset to 100%.
- **Shift+1 / Shift+2** -- fit the whole map / fit the selection.
- **Space+drag** -- pan from anywhere, even over a topic.
- **Drag the background** to pan; **scroll / pinch** to zoom; **Esc** -- exit focus, drill, a walk, or Present.

Files and app

- **Ctrl+O** -- open a `.mmst` / `.json` / `.mmap` file; **Ctrl+S** -- save to the linked file; **Ctrl+Shift+S** -- save as.
- **Ctrl+,** -- Settings and preferences.

Presenting

- **Arrow keys / Space** -- move between slides; **Home** -- first slide.
- **P** -- presenter view (notes, next-up, timer, agenda); **B / W** -- black / white curtain; **Esc** -- exit.

> If you only memorise three things: **Tab** goes deeper, **Enter** stays level, and **Ctrl+K** does anything by name. Everything else you can reach from the toolbar while you learn.

Appendix B -- Format reference

What goes in, what comes out, and what survives the trip. For the day-to-day guidance on *which* format to reach for, see Chapter 6; this appendix is the detail.

Export formats

- **JSON** (native) -- **lossless** . The complete model: topics, notes, markers, images, boundaries, relationships, styling. The format to use whenever you might re-open the map.
- **Markdown** -- the tree as an outline (`#` root, nested bullets). Round-trippable as structure; canvas-only detail (colour, arrows) is naturally dropped.
- **OPML** -- standard outliner interchange. Structure and topics; not canvas styling.
- **FreeMind / Freeplane (`.mm`)** -- the most widely-read mind-map format: topic tree, links, folded state, and notes. The bridge to most other mind-mappers.
- **Mermaid (`.mmd`)** -- the `mindmap` text format you embed in Markdown that renders Mermaid (GitHub, GitLab, Notion, many docs tools).
- **XMind (`.xmind`)** -- the modern (2020+) XMind package: topics, notes, links, tags, floating topics, and relationships.
- **SimpleMind (`.smmx`)** -- the native SimpleMind format: topic tree, notes, web links, relations, and floating topics.
- **MindManager (`.mmap`)** -- topic tree, notes, hyperlinks, stock icons, tags, task info (dates, priority, progress), embedded images, relationships, boundaries, and the two-sided arrangement; the mirror of the importer below.
- **PNG** -- a raster image of the canvas, for slides and chat; plain, @2x, @4x, and transparent variants.
- **SVG** -- a vector image: crisp at any zoom, ideal for high-resolution use.
- **HTML** -- a single self-contained page, openable in any browser.
- **Interactive HTML** -- the same single file, but navigable: the visual map with pan/zoom plus a collapsible, searchable outline.
- **HTML slide deck** -- the walk-through as a standalone, navigable presentation in one self-contained file (an overview slide, then one per branch as its live map image, with speaker notes).
- **PowerPoint (`.pptx`)** -- a real, editable slide deck (an overview slide, then one per branch with its subtree as bullets); a minimal PresentationML package that opens in PowerPoint, Keynote, LibreOffice, and Google Slides.
- **Excel (`.xlsx`)** -- the map as an indented outline worksheet (each topic in the column matching its depth, plus a Notes column); a minimal SpreadsheetML package that opens in Excel, LibreOffice Calc, Numbers, and Google Sheets.
- **Word (`.docx`)** -- the map as an editable outline document (title, indented bulleted topics, notes as italic lines); a minimal Open-XML package that opens in Word, LibreOffice, Pages, and Google Docs.
- **PDF** -- written directly as a real file (fit-to-map, A4, or Letter), or via the browser's print path when you want its options.

- `.mmst` -- not an export but the native *file* format (the lossless JSON under the app's own extension), for keeping a map on disk with autosave writing through to it (Chapter 6).

Import formats

- **JSON** (native) -- the lossless round trip of the JSON export.
- **Markdown** -- any # /bullet outline becomes a map, including **Markmap** -flavoured Markdown (YAML frontmatter plus multi-level headings).
- **OPML** -- outlines from other tools.
- **FreeMind / Freeplane** `.mm` -- topics, links, folded state, and notes.
- **Mermaid** `.mmd` -- `mindmap` text; the hierarchy comes from the indentation.
- **XMind** `.xmind` (2020+) -- topics, notes, web links, and labels (as tags).
- **SimpleMind** `.smmx` -- topic tree, notes, web links, and relations.
- **MindMup** `.mup` -- the JSON maps from the browser-based MindMup.
- **TextBundle / TextPack** (`.textpack`) -- the bundle's Markdown (`text.md`) becomes the map; what Bear, Ulysses, and iA Writer export.
- **iThoughts** `.itmz` , **MindMeister** `.mind` , **Word** `.docx` , **Excel** `.xlsx` -- topic tree (and notes, where the format carries them).
- **MindManager** `.mmap` -- one-way, lossy (see below).
- **Batch import** -- select several files at once to create several maps in one step.

What the `.mmap` importer recovers

The MindManager importer was built from MindManager's published XML schema, not guessed at. From a `.mmap` file it recovers:

- the **topic tree** and all topic text;
- **notes** (the full body), **tags** , and **rich text** (bold / italic / underline, colour);
- **stock icons** , mapped to the closest **emoji** marker;
- **hyperlinks** , **embedded images** , and **attachments** ;
- **relationships** and **boundaries** , with their styling; **callouts** ;
- **task info** -- start and due dates, priority, progress;
- the two-sided arrangement, the **map background** , and explicit per-topic colours, fonts, and shapes;
- **floating topics** .

What it deliberately leaves behind

A few MindManager-only things still don't cross: theme- *inherited* (non-explicit) styling, summary brackets, vector (EMF/WMF) images with no raster fallback, and the project-management layer beyond basic task info -- schedules, durations, resource assignments, Gantt data. The importer **warns** you about what it skipped rather than dropping it silently, and it does a left-behind check for any topic it couldn't reach. Treat the import as a migration -- a clean start in a new tool -- not as a two-way sync.

A practical note on lossless round trips

Only **JSON in and JSON out** is guaranteed to reproduce a map exactly. If a map matters, keep a JSON export (or a whole-library backup, Chapter 7) as the canonical copy, and treat the other formats as *views* of it -- excellent for sharing, not the thing you archive.

Appendix C -- Further reading

This book is deliberately practical -- enough technique to make the tool pay off, no more. If the *why* of visual thinking has caught your interest, here is where the deeper water is.

On mind mapping itself

The modern popular form of the mind map -- a central image, radiating branches, colour and keywords -- was popularised by **Tony Buzan** , who argued that this "radiant" layout mirrors how association actually works better than a linear list does. His books are the canonical starting point, and worth reading even where you end up disagreeing with the stricter rules he proposed. ("Mind Map" is a term associated with the Buzan Organisation; MindMap Studio uses the phrase descriptively for the diagram type and is an independent project.)

On why it works

Chapter 2's claims about the shape have older, deeper roots worth chasing. The limits of **working memory** -- the handful of items you can hold at once, and what happens when a task exceeds it -- are one of the most replicated findings in cognitive psychology; the literature on **cognitive load** carries it into learning and instruction. The value of pairing words with spatial and visual structure is studied as **dual coding** ; the strength of *place* as a retrieval cue is as old as the ancient **method of loci** and as current as research on **spatial memory** . And the idea that thinking improves when it's moved out of the head and onto a rearrangeable surface is the field of **distributed and extended cognition** . None of this is required reading to use a map -- but if you want to know why the technique keeps working, these are the load-bearing walls.

On externalising thought

The broader idea -- that thinking gets better when you get it *out of your head* and onto a surface you can rearrange -- runs through a lot of good writing on note-taking, sketching and "thinking on paper". Look into the literature on **visual note-taking** and **sketchnoting** for the hand-drawn tradition, and the **personal knowledge management** community for the digital one. Mind mapping sits between them: more structured than a sketch, more spatial than an outline.

On structure and decomposition

A mind map is one way to break a big thing into parts. If you find yourself wanting more rigour about *how* to decompose -- mutually exclusive branches, collectively exhaustive coverage, cause-and-effect rather than mere grouping -- the worlds of **issue trees** , **logic trees** and **causal mapping** are the next step out. They trade some of the mind map's freedom for more disciplined structure, and the two skills reinforce each other.

On the tool

MindMap Studio is open source. The application code is licensed under Apache 2.0; this book

under Creative Commons BY-NC 4.0. The repository, the issue tracker, the live app, and a dashboard of the project's own metrics are all linked from mindmap-studio.struktureretsundfornuft.dk . If a chapter here describes something the app no longer does exactly that way, the repository is the source of truth -- and a good place to report the drift.

The shortest possible reading list

If you read only one more thing: pick a real problem you're avoiding, open a blank map, put the problem in the centre, and spend twenty minutes branching it. The technique teaches itself faster than any book can, this one included. The reading is for *after* you've felt it work.