Forests and ranks

The forest

A **forest** is a group of trees — the full scope of knowledge for a single **role**. "Front End Developer" is a forest. Its trees (Testing, Frameworks, JavaScript/TypeScript, Styling/CSS) are the logical slices; the forest is the whole territory.

The default shape: **one forest per role, covering every rank.** One "Front End Developer" forest holds everything from the junior's first skills to the senior's specialisations, rather than a separate forest per seniority level. (A forest per rank also works — but one forest per role is usually more convenient, because a person's whole journey stays on one map.)

Two properties fall out of the structure you already know:

Forests have roots too. The same derivation that gives a tree its roots applies at the forest boundary: prerequisites that sit outside the *whole forest* are what you need before entering the role at all. For a front-end forest, that might be general computer literacy or basic HTML — whatever the forest's lowest skills genuinely require but don't contain. Forest roots are the entry requirement to the role, derived, not guessed — which makes them unusually useful for hiring.

Forests are honest about size. A role is a lot of knowledge, and the forest shows all of it. That can feel intimidating on first view — until you notice that nobody is expected to hold the whole forest. You hold *your* skills at *your* levels, and the next chapter's ranks tell you which part of the forest your current chapter of the journey is about.

The forest is the method's unit of "a role, mapped." Everything after this — ranks, coverage, hiring, reviews — is a way of reading one.

Ranks

A forest maps a role's knowledge; **ranks** map the journey through it. A rank is a seniority level — **junior / mid / senior** by default — and each rank is defined as a concrete set of skills, at given levels, that a person at that rank is expected to hold.

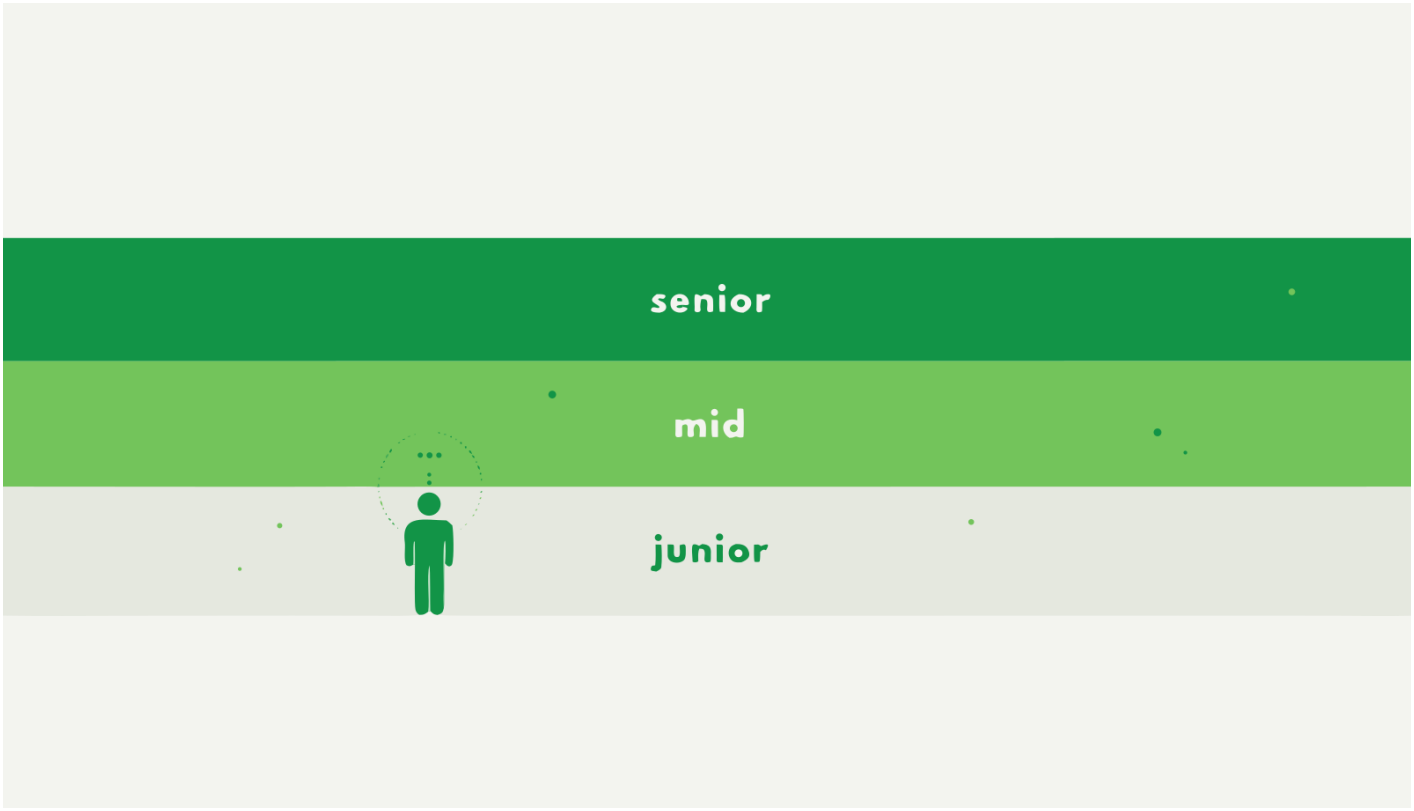
The key subtlety: **ranks attach to a person, not to a thing.** A skill isn't "a senior skill" in nature; the *organisation* decides that holding it (at some level) is part of what senior means here. Setting up ranks is exactly that act: for each forest, choosing which skills — and which levels of them — belong to each rank. Different organisations will slice the same forest differently, and that's fine.

What ranks buy you:

- **A visible "you are here."** A person reads their rank off the forest: they hold the junior set, most of the mid set, and there's the gap.
- **A promotion with a shape.** "Getting to senior" stops being a feeling and becomes a specific, finite list of skills at levels. Both the person and their manager can see it all year, not just at review time.
- **A shared language for expectations.** "Mid" means the same set of skills to everyone looking at the same forest — no private definitions.

On names: junior / mid / senior is the default ladder, but an alternative like **apprentice / journeyman / master** maps onto exactly the same structure. Pick whichever fits the culture; the mechanics don't change.

Ranks are also where the method eventually touches evaluation and pay — a powerful and delicate connection that gets its own treatment in the skill-management chapter.


senior
mid
junior

Roles that share skills

Skills are defined once, organisation-wide — and roles overlap. A front-end developer and a designer both hold "accessibility basics"; a data analyst and a backend developer both hold "SQL." So when you stand two forests side by side, they share ground.

That shared ground is one of the quietly valuable reads in the whole method:

Sideways moves become visible. Someone eyeing a move from QA to backend development doesn't face a mystery — the overlap between the two forests shows what already transfers, and the difference shows exactly what they'd need to pick up, in prerequisite order. A career change becomes a gap analysis instead of a leap of faith.

Hidden versatility surfaces. People often hold skills their current role doesn't showcase. Mapping someone onto a *different* role's forest can reveal they're already halfway into it — useful when a team needs to staff a new kind of work quickly.

The organisation sees its connective tissue. Where many forests share the same skills, those skills are the org's common foundation — worth investing in training for, and worth watching, because a gap there is a gap everywhere at once.

None of this requires extra machinery. It's the payoff of two rules established long before: skills are defined once (so the same skill *is* the same skill in every forest), and prerequisites live on skills (so paths into new territory come pre-drawn). The overlap was always there; the forests just make it visible.

> **[Image — Midjourney prompt]** *watercolor illustration of two distinct forests meeting at a shared grove in the middle, trees from both sides intermingling in the overlap zone, a small path crossing from one forest through the shared grove into the other, two subtly different shades of green blending where they meet, warm linen palette background, hopeful crossing-over mood*

> we need a different prompt, it's too abstract for midjourney

When roles don't fit

Some teams are too varied for fixed ladders. A small agency where everyone wears three hats, a research group, a startup where "the role" changes quarterly — forcing a junior/mid/senior forest onto teams like these produces a map nobody recognises.

The method bends without breaking. Drop the role structure, keep everything else:

Define trees as areas to follow, not rungs to climb. The trees still map real areas of knowledge — but instead of belonging to a role's ladder, they're a landscape on offer.

Let people build their character. Each member chooses which trees to take on, video-game style. One person goes deep on two trees; another spreads across four. The forest stops prescribing a single journey and starts recording each person's chosen one — which, for the right team, is dramatically more motivating.

Manage levels across the team. Coverage, gaps, single points of failure, honest levels, monthly reviews — every team-level read from later chapters still works. You're managing a portfolio of skills instead of progress along ladders, but the map stays true and the discipline stays the same.

What you give up is the crisp promotion mechanics of ranks — "ready for senior" needs some other definition, or the question simply matters less in such teams. What you keep is everything the map is for: knowing what the team can do, seeing risk early, and giving every person a visible, ordered way to grow.

Ranks are a feature of the method, not its foundation. The foundation is skills, honestly held, on a shared map.