# Lean 4 tactic cheatsheet

Last updated: September 30, 2025

If a tactic is not recognized, write `import Mathlib.Tactic` at the top of your file.

| Logical symbol | Appears in goal | Appears in hypothesis |
|---|---|---|
| $\forall$ (for all) | `intro x` | `apply h` or `specialize h x` |
| $\rightarrow$ (implies) | `intro h` | `apply h` or `specialize h1 h2` |
| $\neg$ (not) | `intro h` | `apply h` or `contradiction` |
| $\leftrightarrow$ (if and only if) | `constructor` | `rw [h]` or `rw [← h]` or `apply h.1` or `apply h.2` |
| $\wedge$ (and) | `constructor` | `obtain ⟨h1, h2⟩ := h` |
| $\exists$ (there exists) | `use x` | `obtain ⟨x, hx⟩ := h` |
| $\vee$ (or) | `left` or `right` | `obtain h1\|h2 := h` |
| $a = b$ (equality) | `rfl` or `ext` | `rw [h]` or `rw [← h]` |
| `True` | `trivial` | — |
| `False` | — | `contradiction` |

| Tactic | Effect |
|---|---|
| | **Applying Lemmas** |
| `exact` *expr* | prove the current goal exactly by *expr*. |
| `apply` *expr* | prove the current goal by applying *expr* to some arguments. |
| `refine` *expr* | like `exact`, but *expr* can contain `?_` that will be turned into a new goal. |
| `convert` *expr* | prove the goal by showing that it is equal to the type of *expr*. |
| | **Context manipulation** |
| `have h :` *prop* `:=` *expr* | add a new hypothesis `h` of type *prop*. ⚠ **Do not use for data!** |
| `have h :` *prop* `:= by` *tac* | add hypothesis `h` after proving it using tactics. ⚠ **Do not use for data!** |
| `set x :` *type* `:=` *expr* | add an abbreviation `x` with value *expr*. |
| `clear h` | remove hypothesis `h` from the context. |
| `rename_i x h` | rename the last inaccessible names with the given names. |
| `expose_names` | make all inaccessible names accessible. |
| `show` *expr* | replaces the goal by *expr*, if they are equal by definition. |
| `generalize_proofs` | add all proofs occurring in the goal to the local context. |
| | **Rewriting and simplifying** |
| `rw [`*expr*`]` | in the goal, replace (all occurrences of) the left-hand side of *expr* by its right-hand side. *expr* must be an equality, iff statement or definition. |
| `rw [←`*expr*`]` | ... rewrites using *expr* from right-to-left. |
| `rw [`*expr*`] at h` | ... rewrite in hypothesis `h`. |
| `nth_rw n [`*expr*`]` | rewrite only the *n*-th occurrence of the rewrite rule *expr*. |
| `grw` | Rewrite with inequalities or other relations. |
| `simp` | simplify the goal using all lemmas tagged `@[simp]` and basic reductions. |
| `simp at h` | ... simplify in hypothesis `h`. |
| `simp [*,` *expr*`]` | ... also simplify with all hypotheses and *expr*. |
| `simp only [`*expr*`]` | ... only simplify with *expr* and basic reductions (not with simp-lemmas). |
| `simp?` | ... let Lean speed up `simp` by specifying which lemmas were used. |
| `simp_rw [`*expr1*`, ...]` | like `rw`, but uses `simp only` at each step. |
| `simp_all` | repeatedly simplify the goal and all hypothesis using all hypotheses. |
| `norm_num` | simplify numerical expressions by calculating. |
| `push` *pattern* / `pull` *pattern* | move the constant given by *pattern* inwards / outwards. |
| `norm_cast` | simplify the expression by moving casts ($\uparrow$) outwards. |

| | |
|---|---|
| `push_cast` | push casts inwards. |
| `conv => ` *conv-tac* | apply rewrite rules to only part of the goal. Use `congr`, `skip`, `ext`, `lhs`, `rhs`, . . . to navigate to the desired subexpression. See TPIL. |
| `change ` *expr* | change the current goal to *expr*, if they are equal by definition. |
| `split_ifs` | case split on every occurrence of `if` $h$ `then` *expr* `else` *expr* in the goal. |

**Reasoning with equalities, inequalities, and other relations**

| | |
|---|---|
| `calc ` $a = b$ `:= by` *tac*<br>   `_` $\leq c$ `:= by` *tac*<br>   `_` $< d$ `:= by` *tac* | perform a calculation<br>💡 after writing "`calc _`" Lean can generate a basic calc-block for you.<br>💡 after a `by` shift-click on a subterm in the goal to create a new step. |
| `rfl` | prove the current goal by reflexivity. |
| `symm` | swap a symmetric relation. |
| `trans ` *expr* | split a transitive relation into two parts with *expr* in the middle. |
| `subst h` | if `h` equates a variable with a value, substitute the value for the variable. |
| `ext` | prove an equality in a specified type (e.g. functions). |
| `apply_fun ` *expr* `at h` | apply *expr* to both sides of the (in)equality `h`. |
| `linear_combination` | prove an equality by specifying it as a linear combination of hypotheses. |
| `congr` | prove an equality using congruence rules. |
| `gcongr` | prove an inequality using congruence rules. |
| `positivity` | prove goals of the form $0 < x$, $0 \leq x$ and $x \neq 0$. |
| `bound` | prove inequalities based on the expression structure. |
| `order` | prove purely order-theoretic results. |
| `omega` / `cutsat` | solve linear arithmetic problems over $\mathbb{N}$ or $\mathbb{Z}$. |
| `linarith` | prove linear (in)equalities from the hypotheses. |
| `nlinarith` | stronger variant of `linarith` that can solve some nonlinear inequalities. |

**Reasoning techniques**

| | |
|---|---|
| `exfalso` | replace the current goal by `False`. |
| `by_contra h` | proof by contradiction; adds the negation of the goal as hypothesis `h`. |
| `by_cases h : ` *prop* | case-split on *prop*. |
| `induction n with`<br>`| zero => ` *tac*<br>`| succ n ih => ` *tac* | prove a goal by induction on `n`.<br><br>💡 after writing "`induction n`" Lean can generate the cases for you. |
| `choose f h using ` *expr* | extract a function from a forall-exists statement *expr*. |
| `lift n to ` *type* ` using h` | lifts a variable to *type* (e.g. $\mathbb{N}$) using side-condition `h`. |
| `zify` / `qify` / `rify` | shift an (in)equality to $\mathbb{Z}$ / $\mathbb{Q}$ / $\mathbb{R}$. |

**Searching**

| | |
|---|---|
| `exact?` | search for a single lemma that closes the goal using the current hypotheses. |
| `apply?` | gives a list of lemmas that can apply to the current goal. |
| `rw??` | allows you to shift-click on subterms to get rewrite suggestions. |
| `rw?` | gives a list of lemmas that can be used to rewrite the current goal. |
| `have?  using h1, h2` | try to find facts that can be concluded by using both `h1` and `h2`. |
| `hint` | run a few common tactics on the goal, reporting which one succeeded. |

**General automation**

| | |
|---|---|
| `grind` | general-purpose automation tool with many features. |
| `ring` / `noncomm_ring` / `module`<br>`field` / `abel` / `group` | prove the goal by using the axioms of a commutative ring / ring / module / field / abelian group / group. |
| `aesop` | simplify the goal, and use various techniques to prove the goal. |
| `tauto` | prove logical tautologies. |
| `decide` | run a decision procedure to prove the goal (if it is decidable). |

**Operations on goals/tactics**

| | |
|---|---|
| `swap` | swap the first two goals. |
| `pick_goal ` $n$ | move goal $n$ to the front. |

| | |
|---|---|
| `all_goals` *tac* | run *tac* to all goals. |
| `try` *tac* | run *tac* only if it succeeds. |
| *tac1* ; *tac2* | run *tac1* and then *tac2* (same as putting them on separate lines). |
| *tac1* `<;>` *tac2* | run *tac1* and then *tac2* on all goals generated by *tac1*. |
| <span style="color:red">sorry</span> | admit the current goal. |

| | **Domain-specific tactics** |
|---|---|
| `fin_cases h` | split a hypothesis `h` into finitely many cases. |
| `interval_cases n` | if split the goal into cases for each of the possible values for `n`. |
| `compute_degree` | prove (in)equalities about the degree of a polynomial |
| `monicity` | prove that a polynomial is monic |
| `fun_prop` | prove that a function satisfies a property (continuity, measurability, ...). |
| `measurability` | prove that a set or function is measurable. |
| `filter_upwards [h1, h2]` | Show that an `Eventually` goal follows from the given hypotheses. |
| `slice_lhs`, `slice_rhs` | Focus on a part of a composition in a category. |
| | See the source code for some other category theory tactics. |

**Usage note**
This is a quick overview of the most common tactics in Lean with only a short description. To learn more about a tactic and to learn its precise syntax or variants, consult its docstring or use `#help tactic` *tac*.
This list is not complete, and various tactics are intentionally left out.

**Some useful commands** (Some of these also work as tactics)

| | |
|---|---|
| `#loogle` *query* | 🌐 use Loogle! to find declarations. |
| `#leansearch "`*query*`."` | 🌐 use LeanSearch to find declarations. |
| `#exit` | don't compile code after this command. |
| `#lint` | run linters to find common mistakes in the code *above* this command. |
| `#where` | print current opened namespaces, universes, variables and options. |
| `#min_imports` | print the minimal imports needed for what you've done so far. |
| `#find_home` *name* | find a file that imports all prerequisites of *name*. |
| `#help tactic` *tac* | find information about *tac*. |
| `#help` *category* | list all tactics/commands/attributes/options/notations. |

**Legend**
💡   describes a code action for this tactic.
🌐   requires internet access.