Lean 4 Cheatsheet

In the following table, *name* always refers to a name already known to Lean while *new_name* refers to a new name provided by the user; *expr* designates an expression, for example the name of an object in the context, an arithmetic expression that is a function of such objects, a hypothesis in the context, or a lemma applied to any of these. When one of these words appears twice in the same line, the appearances do not designate the same name or the same expression. For tactics not in the core, the necessary import is written in gray.

Logical symbol	Appears in goal	Appears in hypothesis
∀ (for all)	`intro` *new_name*	`apply` *expr* or `specialize` *name expr*
∃ (there exists)	`use` *expr* `import Mathlib.Tactic.Use`	`cases'` *expr* `with` *new_name new_name* `import Mathlib.Tactic.Cases`
→ (implies)	`intro` *new_name*	`apply` *expr* or `specialize` *name expr*
↔ (if and only if)	`constructor`	`rw [`*expr* `]` or `rw [←` *expr* `]`
∧ (and)	`constructor`	`cases'` *expr* `with` *new_name new_name* `import Mathlib.Tactic.Cases`
∨ (or)	`left` or `right` `import Mathlib.Tactic.LeftRight`	`cases'` *expr* `with` *new_name new_name* `import Mathlib.Tactic.Cases`
¬ (not)	`intro` *new_name*	`apply` *expr* or `specialize` *name expr*

In the left-hand column of the following table, the parts in brackets are optional. The effect of these parts is also in brackets in the right-hand column. It is almost always a matter of specifying that a manipulation, which acts by default on the goal, must be performed rather on a certain hypothesis named *hyp*.

Tactic	Effect
`exact` *expr*	assert that the goal can be satisfied by *expr*
`have` *new_name* : *fact* `import Mathlib.Tactic.Have`	introduce a name *new_name* asserting that *fact* is provable
`unfold` *name* (`at` *hyp*)	unfold the definition of *name* in the goal (or in the hypothesis *hyp*)
`convert` *name* `import Mathlib.Tactic.Convert`	prove the goal by transforming it to an existing proposition *name* and create goals for propositions used in the transformation that were not proved automatically
`convert_to` *expr* `import Mathlib.Tactic.Convert`	transform the goal into the expression *expr* and create additional goals for propositions used in the transformation that were not proved automatically
`rw [` (←) *expr* `]` (`at` *hyp*)	in the goal (or in the hypothesis *hyp*), replace the left-hand side (or the right-hand side, if ← is present) of the equality or equivalence *expr* by the other side
`rw [` *expr* `,` *expr* `]` (`at` *hyp*)	rewrite (parts of) the goal (or the hypothesis *hyp*), using given equalities/equivalences in the given order
`linarith` `import Mathlib.Tactic.Linarith`	prove the goal by a linear combination of hypotheses
`ring` `import Mathlib.Tactic.Ring`	prove the goal by combining the axioms of a commutative (semi)ring
`library_search` `import Mathlib.Tactic.LibrarySearch`	search for a single existing lemma which closes the goal, also using local hypotheses
`by_cases` *new_name* : *expr*	split the proof into two cases depending on whether *expr* is true or false, using *new_name* as name for this hypothesis
`by_contra` *new_name* `import Mathlib.Tactic.ByContra`	start a proof by contradiction, using *new_name* as name for the hypothesis that is the negation of the goal
`contrapose` `import Mathlib.Tactic.Contrapose`	transform a goal of the form *expr* → *expr* into its contrapositive
`push_neg` (`at` *hyp*) `import Mathlib.Tactic.PushNeg`	push negations in the goal (or in the hypothesis *hyp*)
`exfalso` `import Std.Tactic.Basic`	apply the rule *ex falso quod libet* (replaces the current goal by `False`)