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 Lean 4 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` `| intro a b => ...`
→ (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` `| intro a b => ...`

∨ (or)	`left` or `right`
	`import Mathlib.Tactic.LeftRight`
		`cases` *expr* `with`
		`| inl a => ...`
		`| inr b => ...`

¬ (not)	`intro` *new_name*	`apply` *expr* or `specialize` *name expr*

In the left-hand column of the following table, the parts in parentheses are optional. The effect of these parts is also in parentheses in the right-hand column.

Tactic	Effect
`exact` *expr*	assert that the goal can be satisfied by *expr*
`convert` *expr* `import Mathlib.Tactic.Convert`	prove the goal by transforming it to an existing fact *expr* and create goals for propositions used in the transformation that were not proved automatically
`convert_to` *proposition* `import Mathlib.Tactic.Convert`	transform the goal into the goal *proposition* and create additional goals for propositions used in the transformation that were not proved automatically
`have` *new_name* : *proposition* `import Mathlib.Tactic.Have`	introduce a name *new_name* asserting that *proposition* is true; at the same time, create and focus a goal for *proposition*
`unfold` *name* (`at` *hyp*)	unfold the definition of *name* in the goal (or in the hypothesis *hyp*)
`rw [` (`←`) *expr* `]` (`at` *hyp*)	in the goal (or in the hypothesis *hyp*), replace (all occurrences of) the left-hand side (or the right-hand side, if ← is present) of the equality or equivalence *expr* by its other side
`rw [` *expr* `,` *expr* `,` *expr* `]` (`at` *hyp*)	do more rewrites in the given order (again ← possible)
`by_cases` *new_name* : *proposition*	split the proof into two cases depending on whether *expr* is true or false, using *new_name* as name for this hypothesis
`exfalso` `import Std.Tactic.Basic`	apply the rule "False implies anything" a.k.a. "ex falso quodlibet" (replaces the current goal by `False`)
`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
`push_neg` (`at` *hyp*) `import Mathlib.Tactic.PushNeg`	push negations in the goal (or in the hypothesis *hyp*); e.g. change ¬∀ x, *proposition* to ∃ x, ¬*proposition*
`linarith` `import Mathlib.Tactic.Linarith`	prove the goal by a linear combination of hypotheses (includes arguments based on transitivity)
`ring` `import Mathlib.Tactic.Ring`	prove the goal by combining the axioms of a commutative (semi)ring
`simp`	simplify the goal (or the hypothesis *hyp*) by combining some standard equalities and equivalences
`exact?` `import Mathlib.Tactic.LibrarySearch`	search for a single existing lemma which closes the goal, also using local hypotheses