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.

<table>
<tr><th>Logical symbol</th><th>Appears in goal</th><th>Appears in hypothesis</th></tr>
<tr><td>∀ (for all)</td><td>intro new_name</td><td>apply expr or specialize name expr</td></tr>
<tr><td>∃ (there exists)</td><td>use expr
import Mathlib.Tactic.Use</td><td>cases expr with
| intro a b => ...</td></tr>
<tr><td>→ (implies)</td><td>intro new_name</td><td>apply expr or specialize name expr</td></tr>
<tr><td>↔ (if and only if)</td><td>constructor</td><td>rw [expr] or rw [← expr]</td></tr>
<tr><td>∧ (and)</td><td>constructor</td><td>cases expr with
| intro a b => ...</td></tr>
<tr><td>∨ (or)</td><td>left or right
import Mathlib.Tactic.LeftRight</td><td>cases expr with
| inl a => ...
| inr b => ...</td></tr>
<tr><td>¬ (not)</td><td>intro new_name</td><td>apply expr or specialize name expr</td></tr>
</table>

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*
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
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 the other side
rw [*expr* , *expr* , *expr*] (at *hyp*)	do more rewrites in the given order (again ← possible)
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)
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
exact? import Mathlib.Tactic.LibrarySearch	search for a single existing lemma which closes the goal, also using local hypotheses