Lean 4 Cheatsheet

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`
$\to$ (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.mp` or `apply h.mpr`
$\wedge$ (and)	`constructor`	`rcases h with ⟨h1, h2⟩`
$\exists$ (there exists)	`use x`	`rcases h with ⟨x, hx⟩`
$\vee$ (or)	`left` or `right`	`rcases h with h|h`
$a = b$ (equality)	`rfl` (if b is a)	`rw [h]` or `rw [← h]` or `subst h` (if b is a variable)

Tactic	Effect
`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 sub-expressions `?_` that will be turned into new goals.
`convert` *expr*	prove the goal by showing that it is equal to the type of *expr*.
`have h :` *proposition* `:=` *expr*	add a new hypothesis `h` of type *proposition*.
`have h :` *proposition*	... also creates *proposition* as a new goal.
`by_cases h :` *proposition*	create two goals, one where `h` is the hypothesis that *proposition* is true and one where `h` is the hypothesis where it is false.
`exfalso`	replace the current goal by `False`.
`by_contra h`	start a proof by contradiction, where `h` is the hypothesis that the current goal is false.
`push_neg` `push_neg at h`	push negations into quantifiers and connectives in the goal (or in `h`); e.g. change ¬ ∀ `x, P x` to ∃ `x,` ¬ `P x`.
`symm`	swap a symmetric relation.
`trans` *expr*	split a transitive relation into two parts with *expr* in the middle.
`congr`	prove an equality using congruence rules.
`gcongr`	prove an inequality using congruence rules.
`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 or if and only if statement.
`rw [←`*expr*`]`	... rewrites using *expr* from right-to-left.
`rw [`*expr*`] at h`	... rewrite in hypothesis `h`.
`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*`]`	... do not simplify with all standard lemmas, only with *expr*.
`simp?`	... generate a `simp only [...]` tactic that applies the same simplifications.
`simp_rw [`*expr1*`,` *expr2*`]`	like `rw`, but uses `simp only` at each step.
`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?`	gives a list of lemmas that can be used to rewrite the current goal.
`linarith`	prove linear (in)equalities from the hypotheses.
`ring` / `noncomm_ring` `abel` / `group`	prove the goal by using the axioms of a commutative ring / ring / abelian group / group.
`aesop`	simplify the goal, and use various techniques to prove the goal.
`tauto`	prove certain goals using first-order logic.