just Command runner

START HERE

`just --init`
Create a starter justfile

`just --list`
Show recipes

`just`
Run the first recipe

`just --dry-run <recipe>`
Print, don't run

`just <recipe>`
Run a recipe

`just --choose`
Pick with fzf

`just <recipe> <arg>`
Pass arguments

`just --fmt`
Tidy the justfile

justfile found upwards from cwd → `just <recipe>` → **Dependencies** run first, once each → `then` → **Recipe lines** each in its own shell → `cwd = justfile dir` → **Result** stops at first failure

CONCEPTS

just is a command runner, not a build system: recipes always run, no file timestamps are checked. Indent recipe bodies consistently with spaces or tabs.

Every recipe line runs in a fresh shell, so cd and shell variables do not carry to the next line. Use a shebang recipe or && for multi-step state.

just searches parent directories for the justfile and runs recipes in the justfile's directory, so it works from any subfolder.

RUNNING

`just`
Run the default (first) recipe

`just <recipe> [<args>...]`
Run one or more recipes in order

`just --list / just -l`
List recipes with doc comments

`just --dry-run / just -n`
Print recipe lines without running them

`just --show <recipe> / just -s <recipe>`
Print a recipe's source

`just --init`
Write a starter justfile in the project root

`just --choose`
Pick recipes with a chooser (fzf by default)
Set JUST_CHOOSER or pass --chooser to use another picker; recipes needing arguments are not offered.

`just --fmt`
Rewrite the justfile in canonical style

`just <var>=<value> <recipe>`
Override a variable for this run
--set <var> <value> is the long form.

`just --yes <recipe>`
Auto-answer [confirm] prompts

`just --quiet / just -q`
Suppress all output

`just --justfile <file> / just -f <file>`
Use a specific justfile

`just --global-justfile / just -g`
Use the global justfile

RUNNING (cont.)

`just --edit / just -e`
Open the justfile in $VISUAL or $EDITOR

`just --command <cmd> / just -c <cmd>`
Run a command with the justfile's env and .env loaded

`just --evaluate [<var>]`
Print variables with their values

`just --completions <shell>`
Print a shell completion script

RECIPE SYNTAX

A recipe

```
# doc comment shown by --list
build:
    cargo build --release
```

`<name>: <dep> <dep>`
Run dependencies before the recipe

`<name>: <dep> && <after>`
Run a dependency after the recipe body

`<name>: (<dep> "<arg>")`
Pass an argument to a dependency

`@<name>:`
Don't echo this recipe's lines

`@<command>`
Don't echo one line

`-<command>`
Keep going if this line fails

`{{<var>}}`
Insert a variable or expression into a line

`<name> <param>:`
Required parameter

`<name> <param>=<default>:`
Parameter with a default

`<name> +<files>:`
Variadic: one or more values

`<name> *<flags>:`
Variadic: zero or more values

`<name> $<param>:`
Export a parameter as an env var

`alias <short> := <recipe>`
Another name for a recipe

List recipes when run bare

```
default:
    @just --list
```

`[arg("<param>", long="<opt>")]`
Take a parameter as a --option

VARIABLES

`<name> := "<value>"`
Define a variable

`export <NAME> := "<value>"`
Export to recipe environments

VARIABLES (cont.)

`<name> := `<command>``
Capture a command's output

`<a> + <b>`
Concatenate strings

`<dir> / <file>`
Join path parts with a slash

`if <a> == <b> { <x> } else { <y> }`
Conditional expression
Also !=, =~ (regex match) and &&, ||.

`'<text>'`
Literal string, no escapes

`"<text>"`
String with \n \t \" escapes

`set dotenv-load`
Load .env into the environment

`$<NAME>`
Read an env var inside a recipe line

`env("<NAME>", "<default>")`
Env var with fallback

`env("<NAME>")`
Env var, error if unset

`justfile_directory()`
Directory of the justfile

`invocation_directory()`
Directory just was started from

`path_exists(<p>)`
True if the path exists

`require("<tool>")`
Path to a tool, error if missing

SETTINGS

`set shell := ["bash", "-euo", "pipefail", "-c"]`
Choose the shell for lines and backticks

`set dotenv-path := "<file>"`
Load a specific env file (must exist)

`set export`
Export all variables

`set positional-arguments`
Pass args as $1, $2 ... too

`set quiet`
Don't echo recipe lines

`set fallback`
Look in parent directories if a recipe is missing

ATTRIBUTES

`[private]`
Hide from --list (also names starting with _)

`[confirm]`
Ask before running

`[confirm("<prompt>")]`
Ask with a custom prompt

ATTRIBUTES (cont.)

`[group("<name>")]`
Group recipes in --list

`[doc("<text>")]`
Set the --list description

`[no-cd]`
Run in the caller's directory

`[working-directory("<dir>")]`
Run in a given directory

`[linux] / [macos] / [windows] / [unix]`
Define the recipe only on that OS

`[parallel]`
Run dependencies concurrently

SCRIPTS & MODULES

Shebang recipe

```
stats:
    #!/usr/bin/env python3
    import sys
    print(sys.version)
```

Script recipe

```
[script("python3")]
stats:
    import sys
    print(sys.version)
```

`import "<file>"`
Include another justfile's recipes

`mod <name>`
Load <name>.just or <name>/justfile as a module

`just <mod> <recipe> / just <mod>::<recipe>`
Run a module's recipe

Shebang recipes run from a temp file, so the interpreter must be installed; their lines are not echoed.

RECIPES

OS-specific recipe

```
[linux]
open:
    xdg-open http://localhost:8080

[macos]
open:
    open http://localhost:8080
```

Build with a version from git

```
version := `git describe --tags --always`

build:
    docker build -t app:{{version}} .
```

Guard a dangerous recipe

```
[confirm("Deploy to production?")]
deploy env="staging": test
    ./deploy.sh {{env}}
```