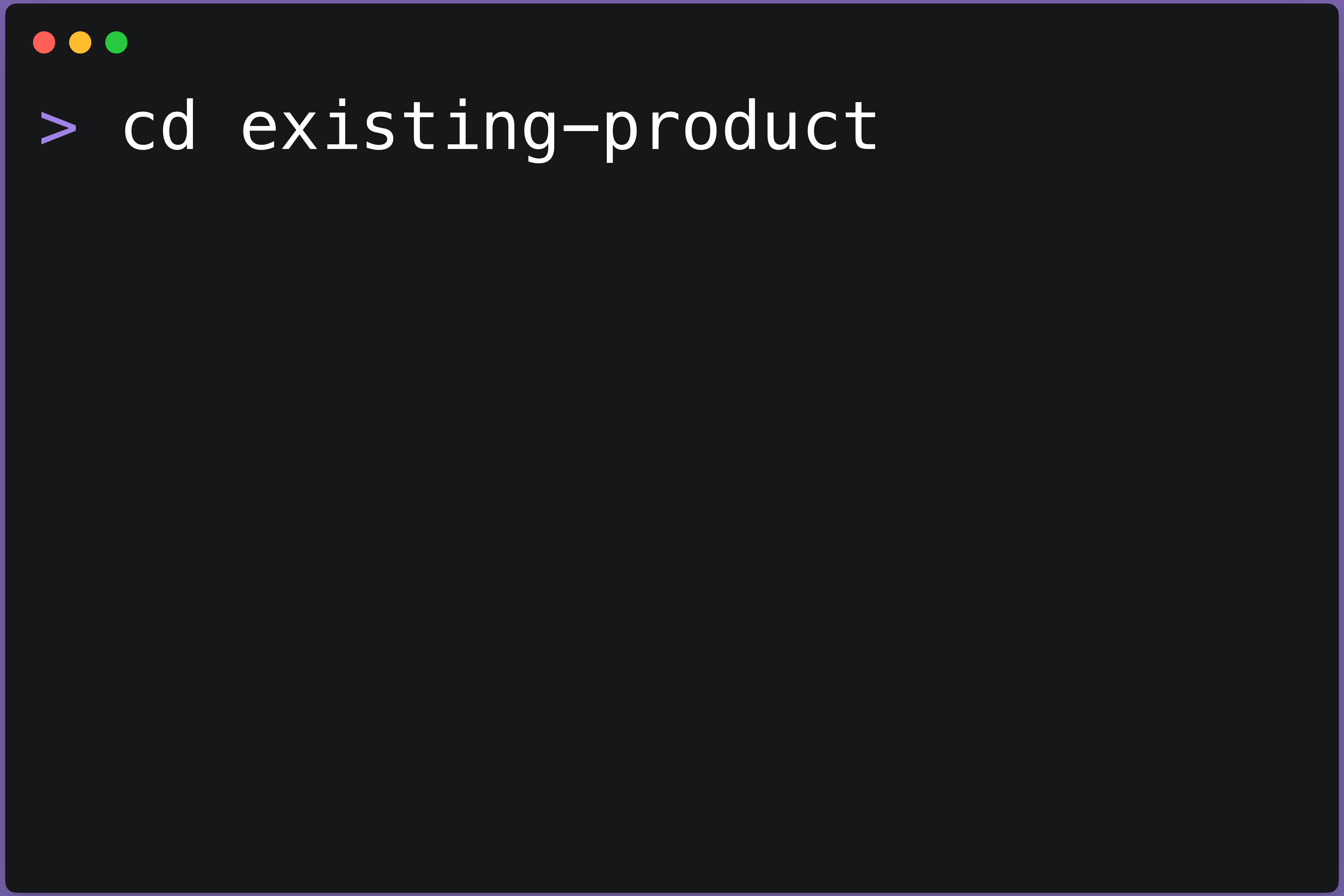
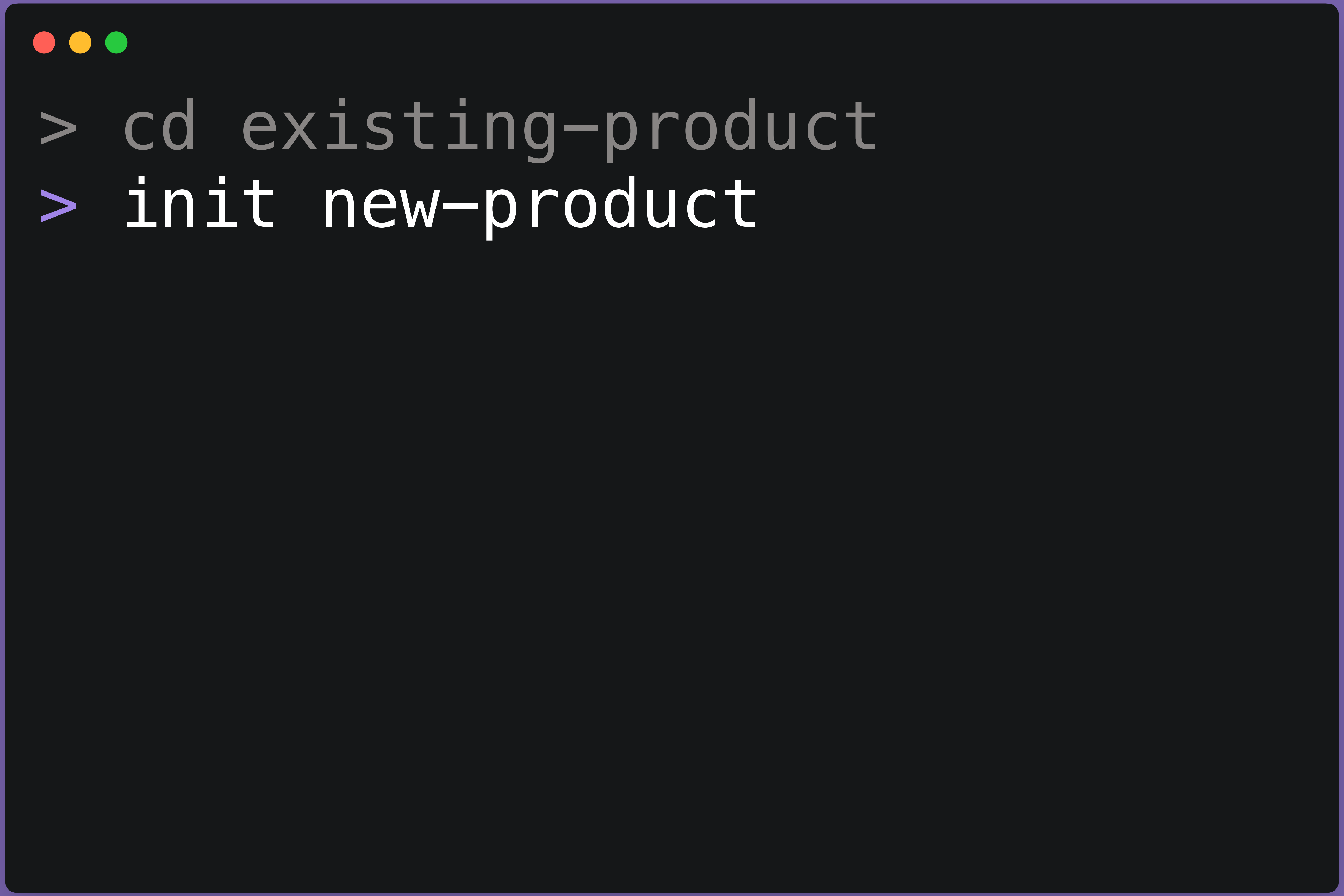
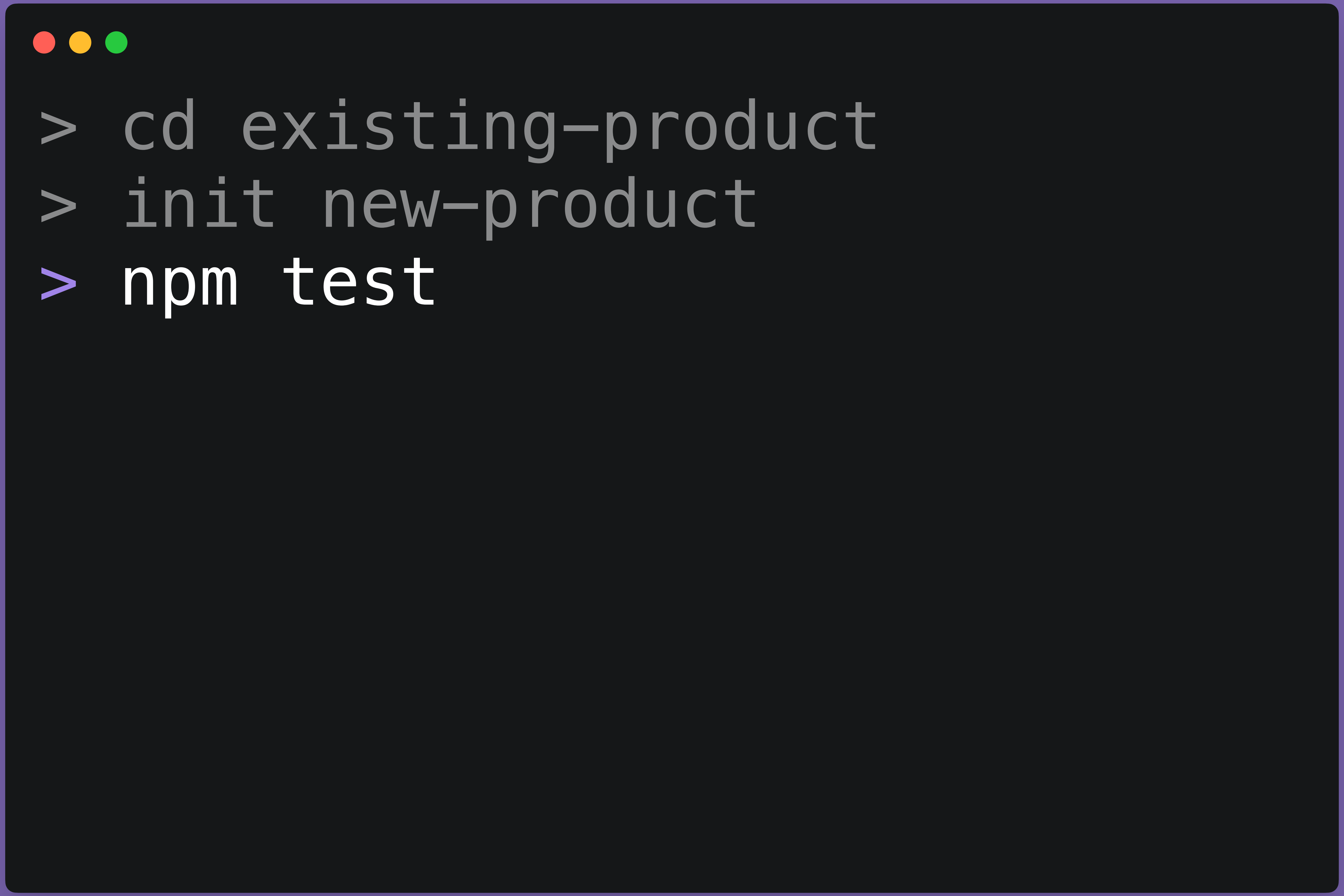
Testing *your* testing strategy



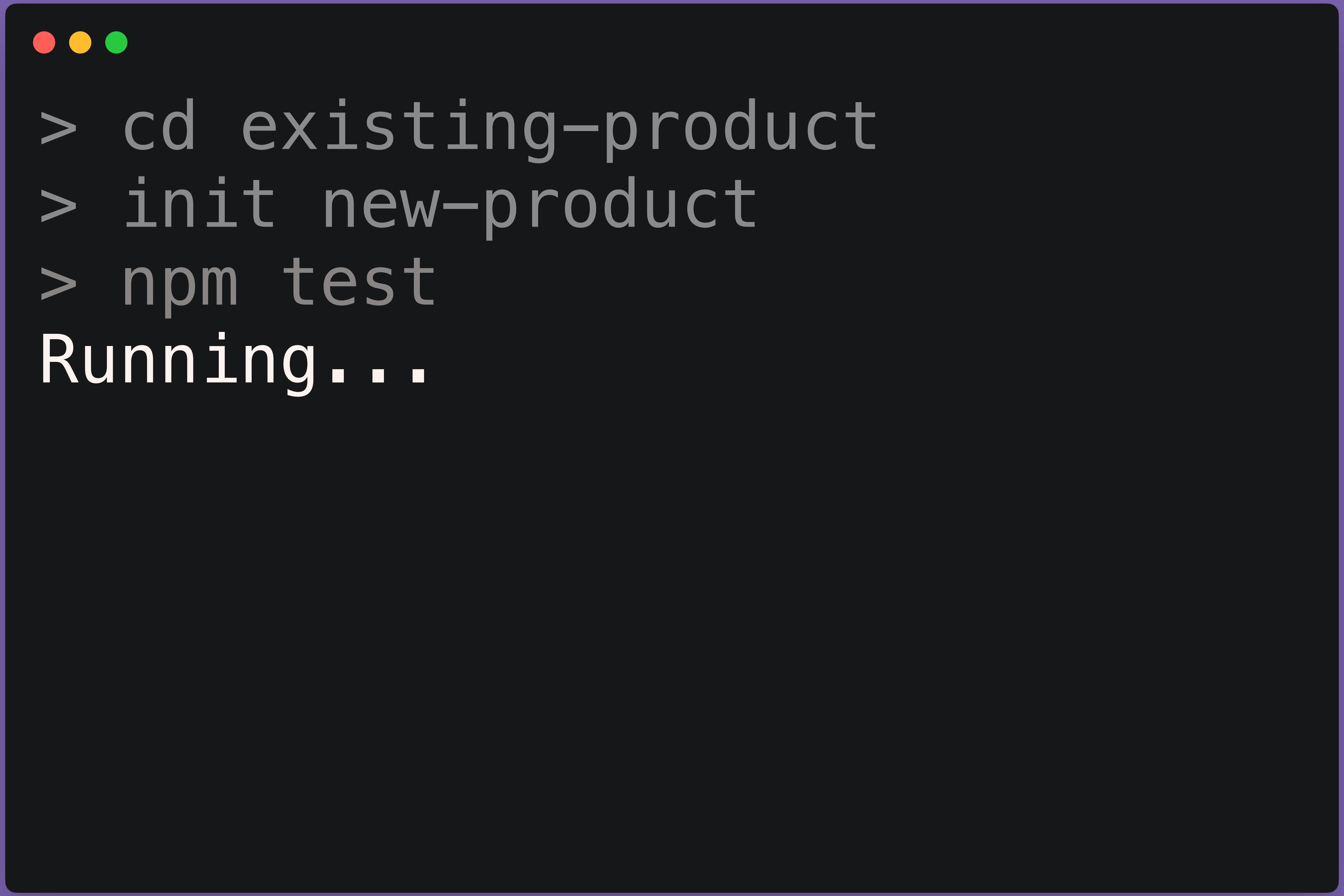
```
> cd existing-product
```



```
> cd existing-product
> init new-product
```



```
> cd existing-product
> init new-product
> npm test
```



```
> cd existing-product
> init new-product
> npm test
Running...
```

“Why do I need to write tests when I might have to rewrite them in a few weeks anyway?

“ **This approach worked for the other project. We can’t ship features without tests!**

Testing *your* testing strategy

Rethinking ~~Testing~~ *your* testing strategy

Trent Willis

Staff Software Engineer, Netflix
Former QUnit project lead
@trentmwillis

Expectation vs. Reality of Testing

Many applications remain under-tested* even though testing is widely regarded as a best practice

*according to their authors

Why testing *feels* inefficient

Why testing *feels* inefficient:

1. Unnecessarily high expectations

Popular testing wisdom advocates for multi-layered approaches for all projects

There is an (over)abundance of testing options

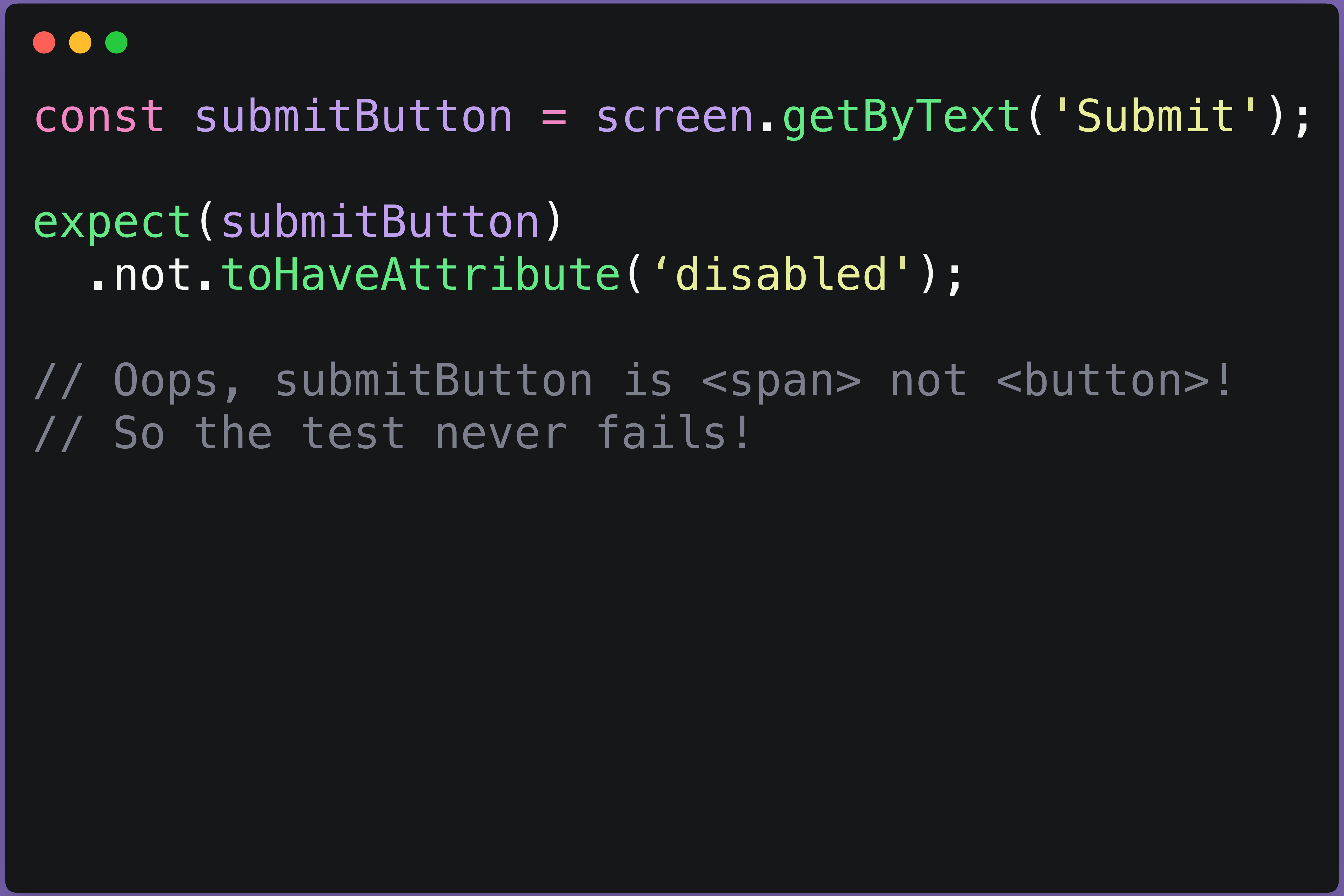
Why testing *feels* inefficient:

1. Unnecessarily high expectations
2. Underdeveloped skills

Learning to write good (i.e., efficient and helpful) tests is a skill that must be developed

Why testing *feels* inefficient:

1. Unnecessarily high expectations
2. Underdeveloped skills
3. Unhelpful tests



```
const submitButton = screen.getByText('Submit');

expect(submitButton)
  .not.toHaveAttribute(‘disabled');

// Oops, submitButton is <span> not <button>!
// So the test never fails!
```

Why testing *feels* inefficient:

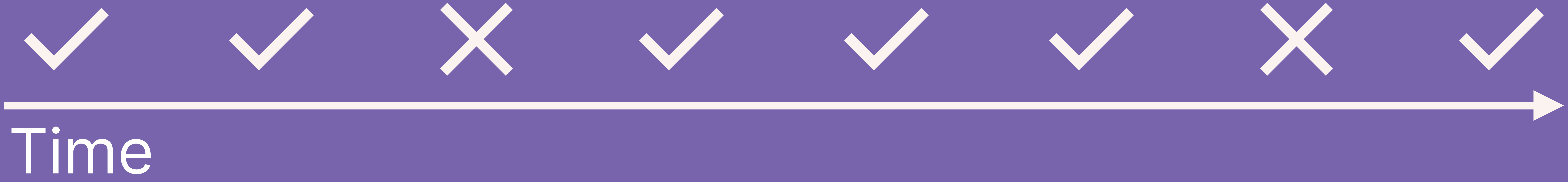
1. Unnecessarily high expectations
2. Underdeveloped skills
3. Unhelpful tests

The value of testing is in quality not quantity

How do you create a valuable testing strategy?

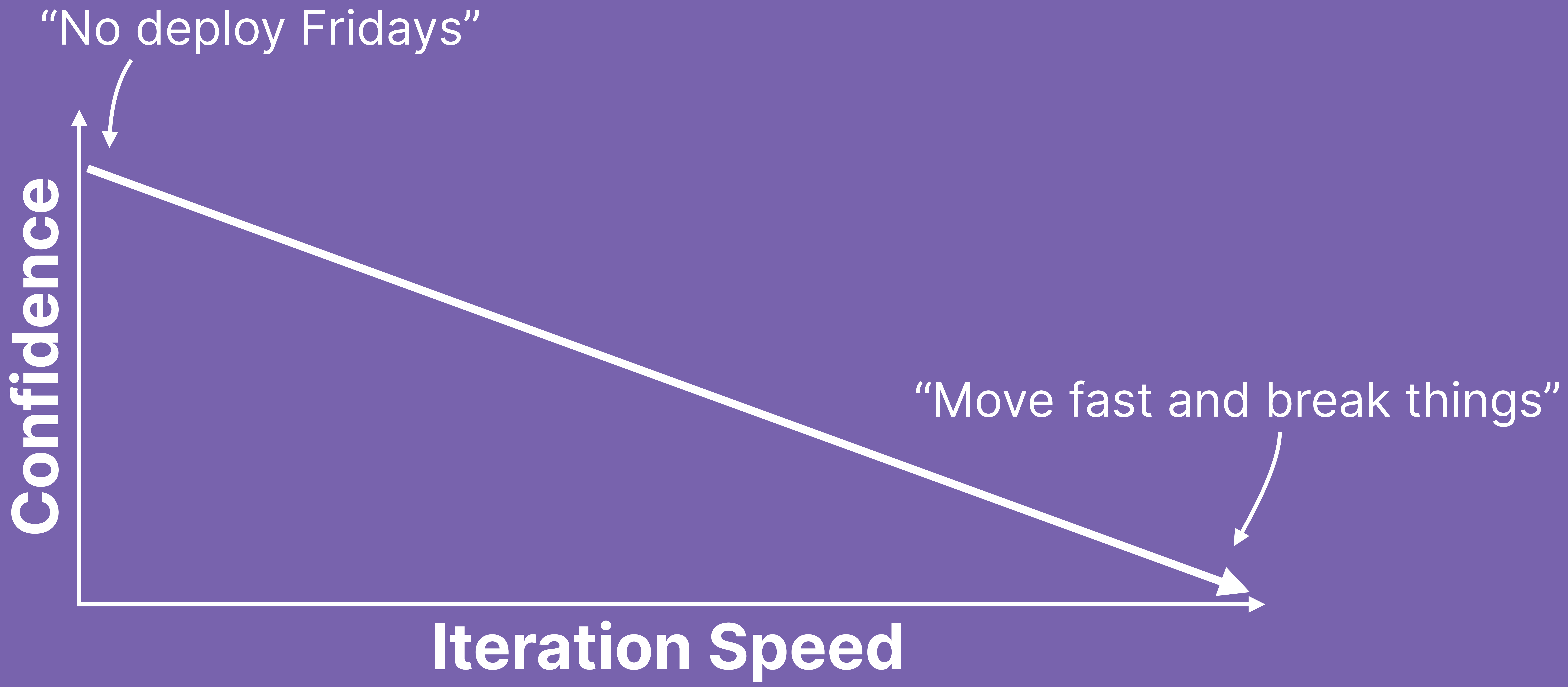
Start by defining your goals

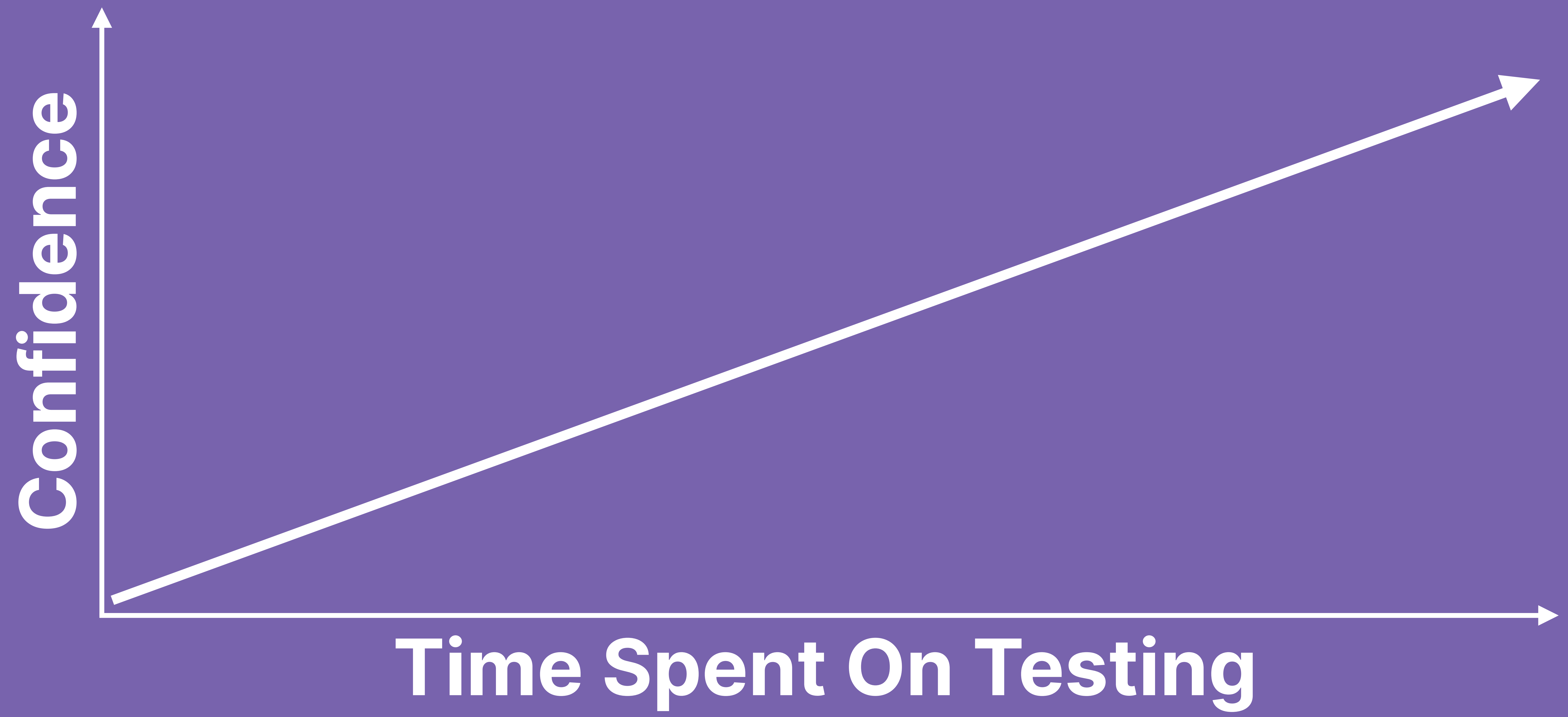
✓ Requirements met


Time

The best way to *not* break an application is to *not* change it, but we change software to make it *better*

Iteration Speed vs. Confidence


"No deploy Fridays"
Confidence
"Move fast and break things"
Iteration Speed


Confidence
Time Spent On Testing

The goal of a testing strategy should be to *optimize* the amount of confidence you get for the time you invest in testing

How much confidence do you need?

How much confidence do you need?

1. What stage of development are you in?
2. What are your users’ expectations?
3. What are the market expectations?

How much confidence do you need?

1. **What stage of development are you in?**
2. What are your users’ expectations?
3. What are the market expectations?

As a product “matures”, you need more confidence

Your confidence level should mirror your confidence that requirements are stable

How much confidence do you need?

1. What stage of development are you in?
2. **What are your users' expectations?**
3. What are the market expectations?

The cost to access and criticality of a product are directly related to how much confidence you need

You can codify user expectations with service-level agreements (*SLAs*)

How much confidence do you need?

1. What stage of development are you in?
2. What are your users’ expectations?
3. **What are the market expectations?**

Good understanding of your product market expectations avoids *over–* **or** *under*–**investing in your testing strategy**

Example

Early development stage +
Low user expectations +
New market space =
Iteration Speed > Confidence

Evaluate your options for building confidence

Non-Testing Options

Non-Testing Options
Static Analysis

Linting
Type checking

Non-Testing Options

Static Analysis

Ideological Changes

Reduce size + complexity
Encapsulation
SOLID (Single Responsibility)
Presentational + Container

Non-Testing Options

Static Analysis
Ideological Changes
Processes

Code reviews
Easy + fast deployments
Canaries / AB Tests
Observability + monitoring

Non-Testing Options

Static Analysis
Ideological Changes
Processes

Non-Testing Options can build confidence without *significantly* impacting iteration speed, especially with changing requirements

“Actual” Tests

Unit, end-to-end, integration, smoke, acceptance, regression, functional, behavioral, etc.

Tests that validate a user flow end-to-end
Tests that validate an isolated unit of functionality

end-to-end tests

unit tests

end-to-end tests

Mimic user flow

Complicated, slow, flakey

Confidence a user flow works

unit tests

end-to-end tests
Mimic user flow
Complicated, slow, flakey
Confidence a user flow works

unit tests
Isolated functionality
Simple, fast, stable
No guarantee user flow works

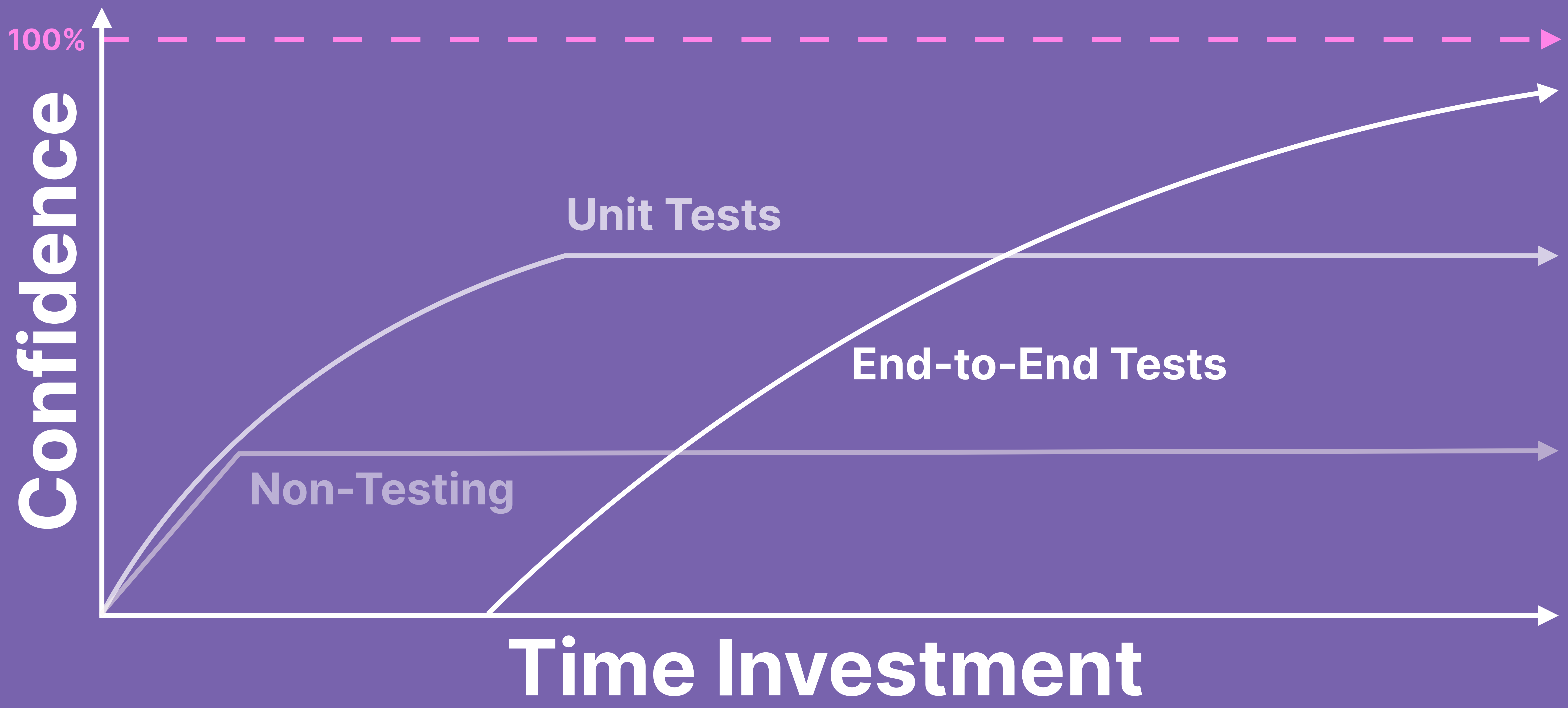
end-to-end tests
Mimic user flow
Complicated, slow, flakey
Confidence a user flow works

unit tests
Isolated functionality
Simple, fast, stable
No guarantee user flow works

Ideally, **end-to-end tests are more valuable**
Really, **unit tests are often a better value**

Doing *one* **or the other** *well* **is better than doing both** *poorly*

Valuable tests are those that catch issues


100%
Confidence
Unit Tests
End-to-End Tests
Non-Testing
Time Investment

Put it all together

Iteration Speed > Confidence

What can you do within the time constraints?

Ex: Barely enough time to deliver product as-is
Static analysis (linting + type checking)
Processes (fast deployments)
Add unit/end-to-end tests later

Ex: In between Alpha and Beta phases

Ideological changes (refactor monolithic modules)
Add unit tests for complex data logic

Confidence > Iteration Speed

What options give us the confidence needed?

Ex: Read-only app
End-to-end (visual regression) tests
Static analysis (linting + type checking)

Ex: Highly configurable dashboard app

Unit tests (covering every option)
End-to-end tests (for basic combinations of options)
Ideological changes (small modules)
Processes (fast deployments, code reviews)
Static analysis (linting + type checking)

Balance your time investment with the level of confidence you need at a *given period* in time

Strategy should be revisited periodically

Your mix of choices can/should change over time

TL;DL (Too Long; Didn’t Listen)

TL;DL (Too Long; Didn’t Listen)

Define your goals
Evaluate your options
Make choices based on your goals

A custom testing strategy benefits your users, company, and team

You don’t need to reinvent the wheel, but you also don’t need to use the same wheels as everyone else