NPM@5

NOT JUST SPEED; BUT ALSO SPEED

OK BUT WHAT'S DIFFERENT

» It goes fast 🏃‍♀️

» Intro to Structure of the CLI 🌯

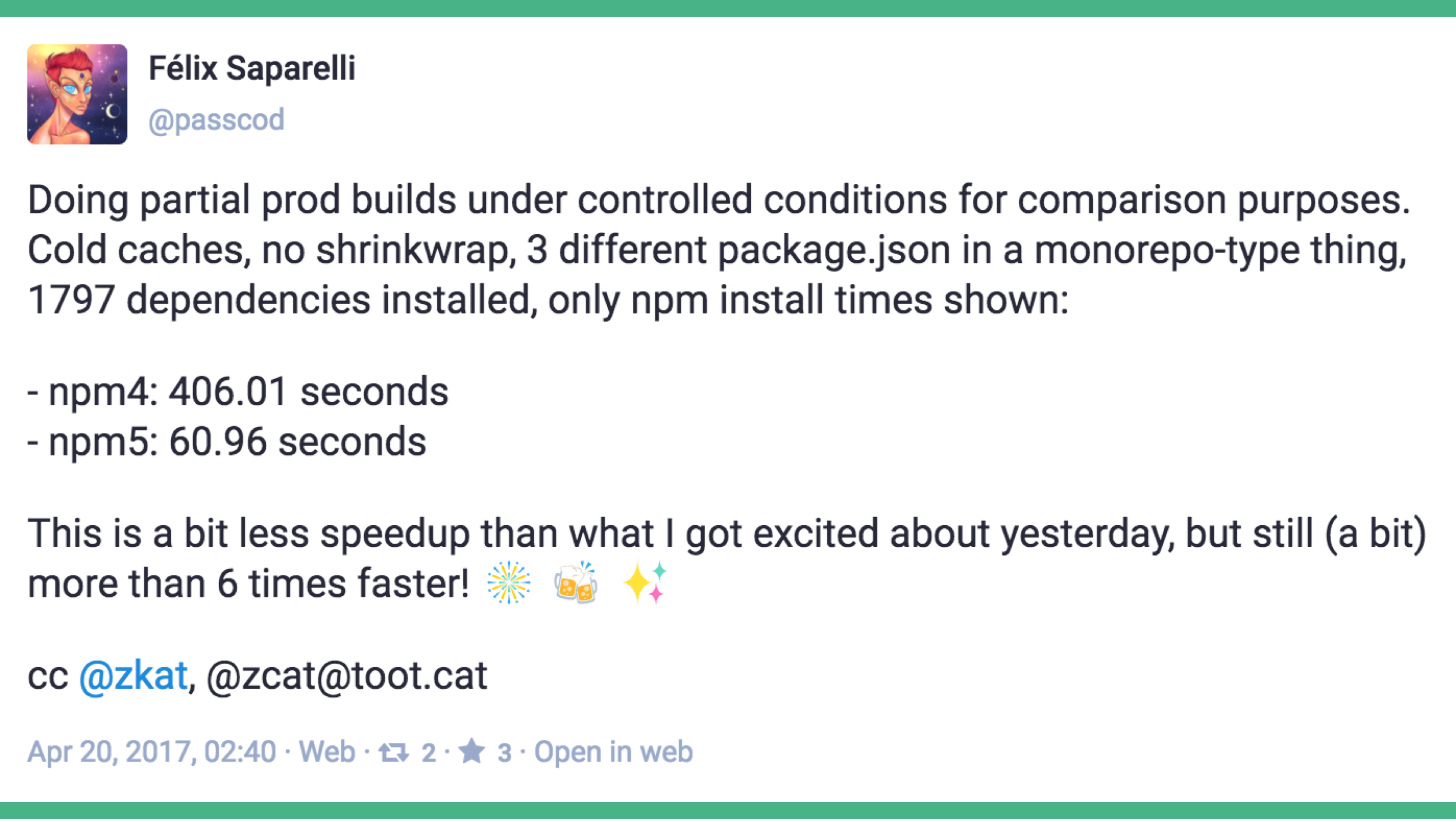
» Why the Cache Rewrite™

Talk is gonna talk about npm@5 speeds

BENCHMARKS: NPM-WEBSITE W/ SHRINKWRAP

TOOL	COLD CACHE	WARM CACHE	WARM CACHE + SCRIPTS
npm@4	98.894s	144.191s 🤔	167.817s
npm@5	65.823s	41.406s	85.981s
yarn@0.22.0	41.73s	30.41s	73.87s

Straight to the point: here's the summary of current benchmarks


Félix Saparelli
@passcod

Doing partial prod builds under controlled conditions for comparison purposes. Cold caches, no shrinkwrap, 3 different package.json in a monorepo-type thing, 1797 dependencies installed, only npm install times shown:

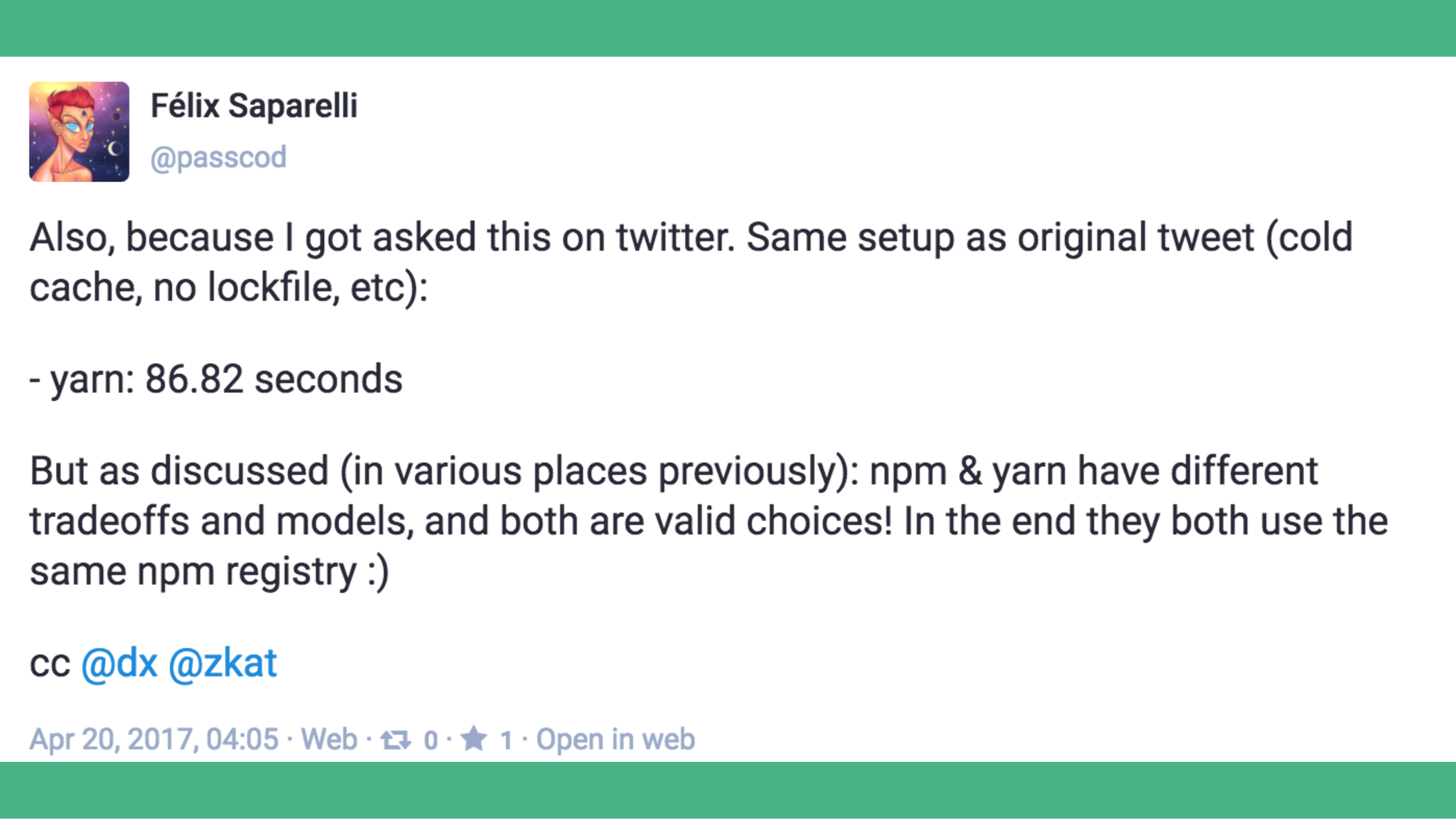
- npm4: 406.01 seconds
- npm5: 60.96 seconds

This is a bit less speedup than what I got excited about yesterday, but still (a bit) more than 6 times faster! 🎇 🍻 ✨

cc @zkat, @zcat@toot.cat

Apr 20, 2017, 02:40 · Web · 2 · 3 · Open in web

Brave volunteer tried out npm5 beta



Félix Saparelli
@passcod

Also, because I got asked this on twitter. Same setup as original tweet (cold cache, no lockfile, etc):

- yarn: 86.82 seconds

But as discussed (in various places previously): npm & yarn have different tradeoffs and models, and both are valid choices! In the end they both use the same npm registry :)

cc @dx @zkat

Apr 20, 2017, 04:05 · Web · 0 · 1 · Open in web

This was really unexpected, and will not be the norm, but whoa

🔥 566% 🔥

NPM@5 VS NPM@4

41%

NPM@5 VS YARN

TL;DR

Heads up - get the timing right for switching the next few slides

TL;DR

NPM@5 IS

TL;DR

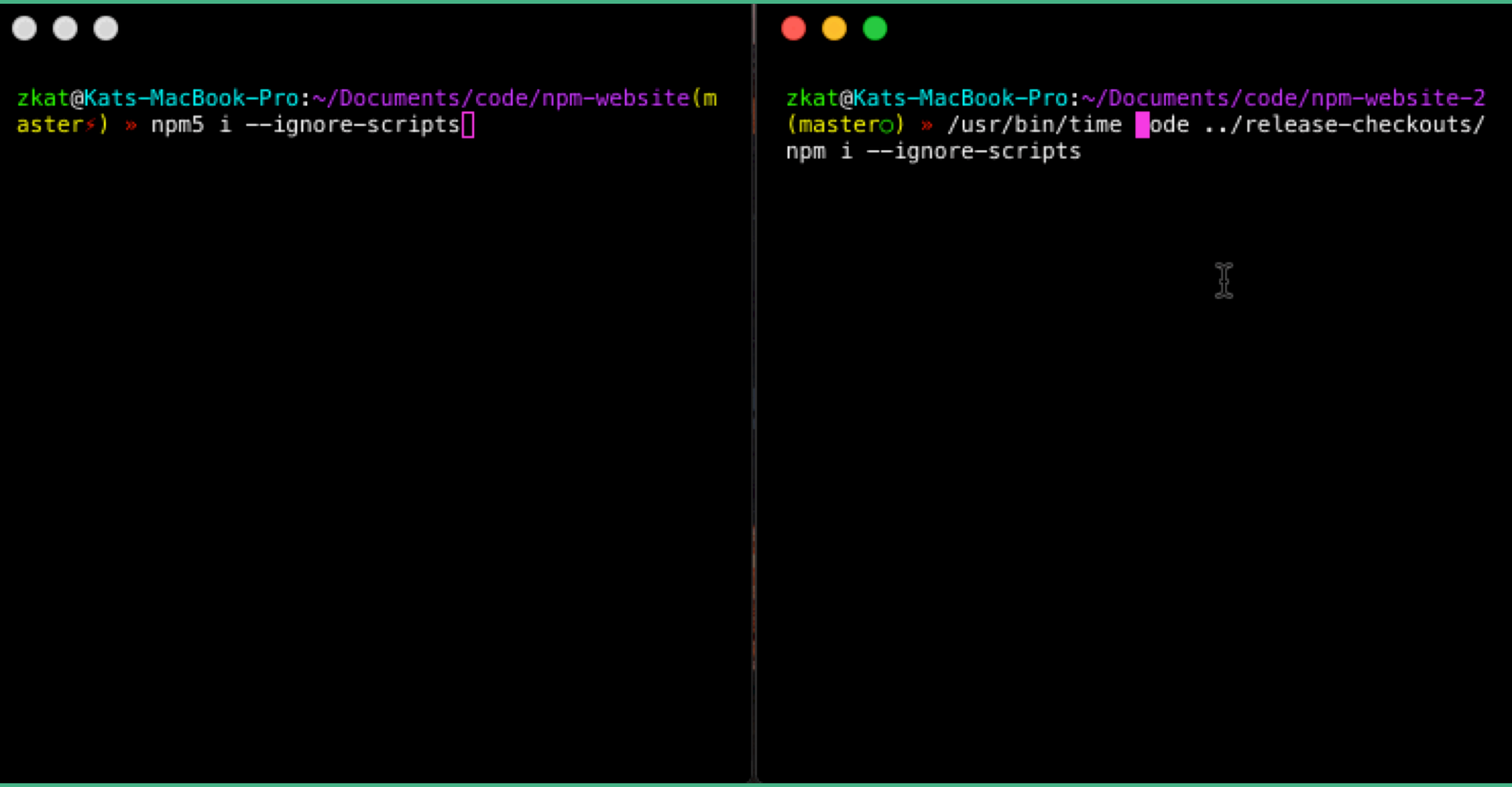
NPM@5 IS REALLY

TL;DR

NPM@5 IS REALLY @$%^ING

TL;DR

NPM@5 IS REALLY #$%^ING FAST



```
zkat@Kats-MacBook-Pro:~/Documents/code/npm-website(m
aster⚡) » npm5 i --ignore-scripts
```

```
zkat@Kats-MacBook-Pro:~/Documents/code/npm-website-2
(master○) » /usr/bin/time node ../release-checkouts/
npm i --ignore-scripts
```

But really, just look here. This is npm-website install. Left is npm5, right is npm4. (wait for npm5) It should tell you something that I probably shouldn't make you sit here and wait for npm@5 to finish. Let's move on.

BUT WHY? 🤷🏽‍♀️

That's cool, but there's a reason, right? Did we break it?

INTERMISSION: INSTALLATION STEPS

Since npm@3, we've had an installer that does entire tasks in individual phases. Let's look at those steps in the pipeline.

INSTALLER PHASES

- read current node_modules (5%)[1]
- fetch packuments (corgis!🐕) (10%)
- fetch tarballs (pkg-1.2.3.tar.gz) (20%)
- extract tarballs (30%)
- run scripts (30%)

[1] I literally just estimated these %s. Not precise.

These is a 10k foot view of the steps the CLI takes. The %s are rough for the sake of illustrating scale and probably fairly wrong in practice, and will change a long depending on many different conditions.

INSTALLER PHASES NPM@5 IMPROVES

- ~~read current node_modules~~ (5%)
- fetch packuments (corgis!🐕) (10%)
- fetch tarballs (pkg-1.2.3.tar.gz) (20%)
- extract tarballs (30%)
- ~~run scripts~~ (30%)

npm@5 only really affects two of these phases

SECRET SAUCE

» cacache - really fast, reliable, secure cache

» make-fetch-happen - HTTP requests + cacache

» pacote - corgi🐕 and tarball fetch API

npm@5 improved these steps with 3 fresh new projects

CACACHE

» content-addressable

» robust/fault-tolerant

» safe, lockless concurrency

» 100% verified on insert and extract

» really fast

» caches corgis and tarballs to disk + memory

MAKE-FETCH-HAPPEN

» get in loser, we're doing http 😎

» retries on failures

» streaming and/or bulk support

» standard http caching w/ cacache under the hood

» ^- means registry can control cache settings

PACOTE

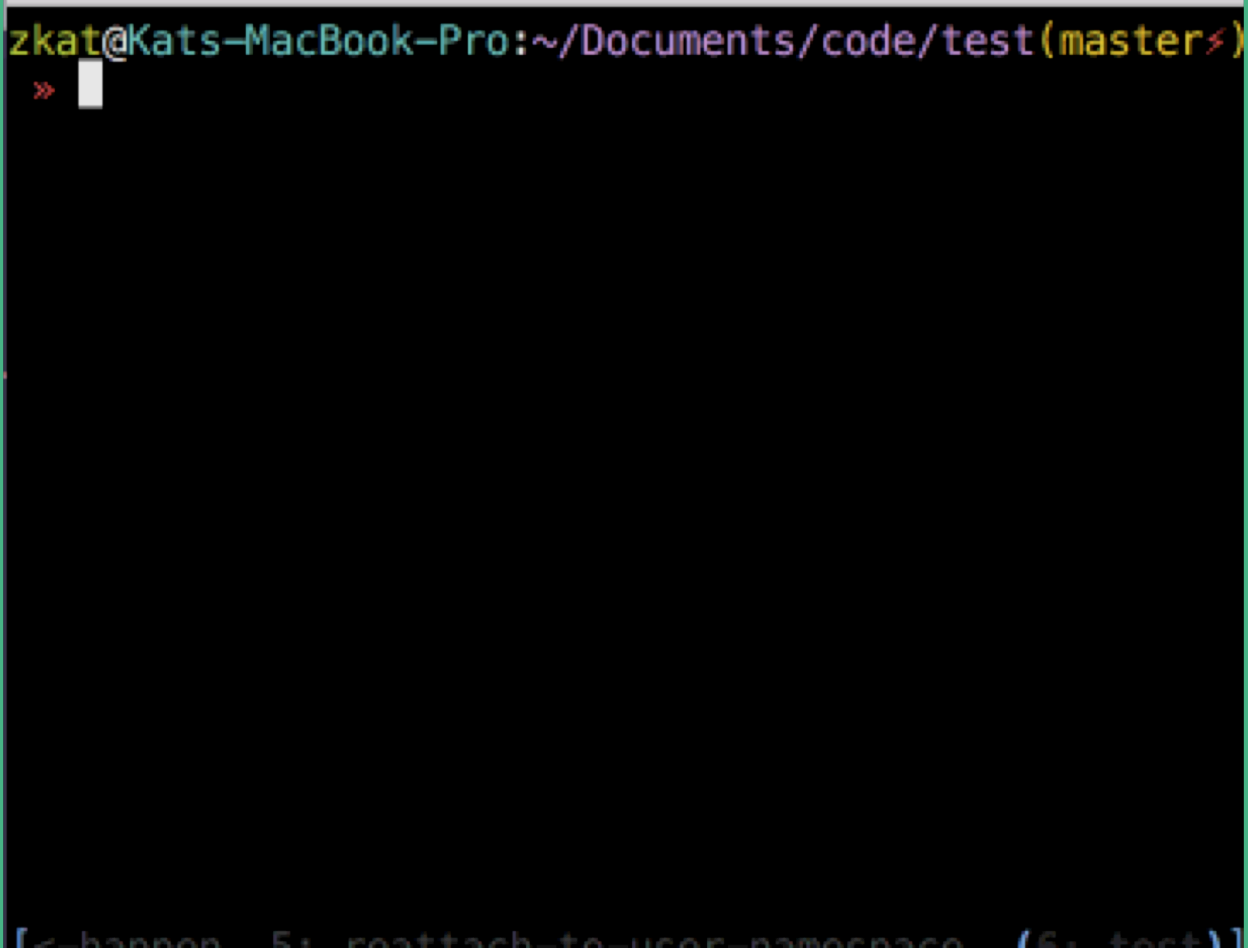
- "package" in Portuguese - "pah-COH-chay"
- puts it all together
- pacote.manifest('foo@latest') -> corgi
- pacote.extract('foo@latest', './here') -> tar data

SHINY NEW FEATURES

But this is not just a story about speed. These projects helped add a bunch of new features to npm, and laid down essential groundwork for work we're going to do right after npm@5 comes out.

AUTOMATIC OFFLINE MODE

This mode means you no longer have to do anything special to get your install working when you don't have internet. It'll just happen, and it'll happen fast. npm@5 adds a few options for controlling how caching works, but cache-min and cache-max, if you're familiar with those, are deprecated.

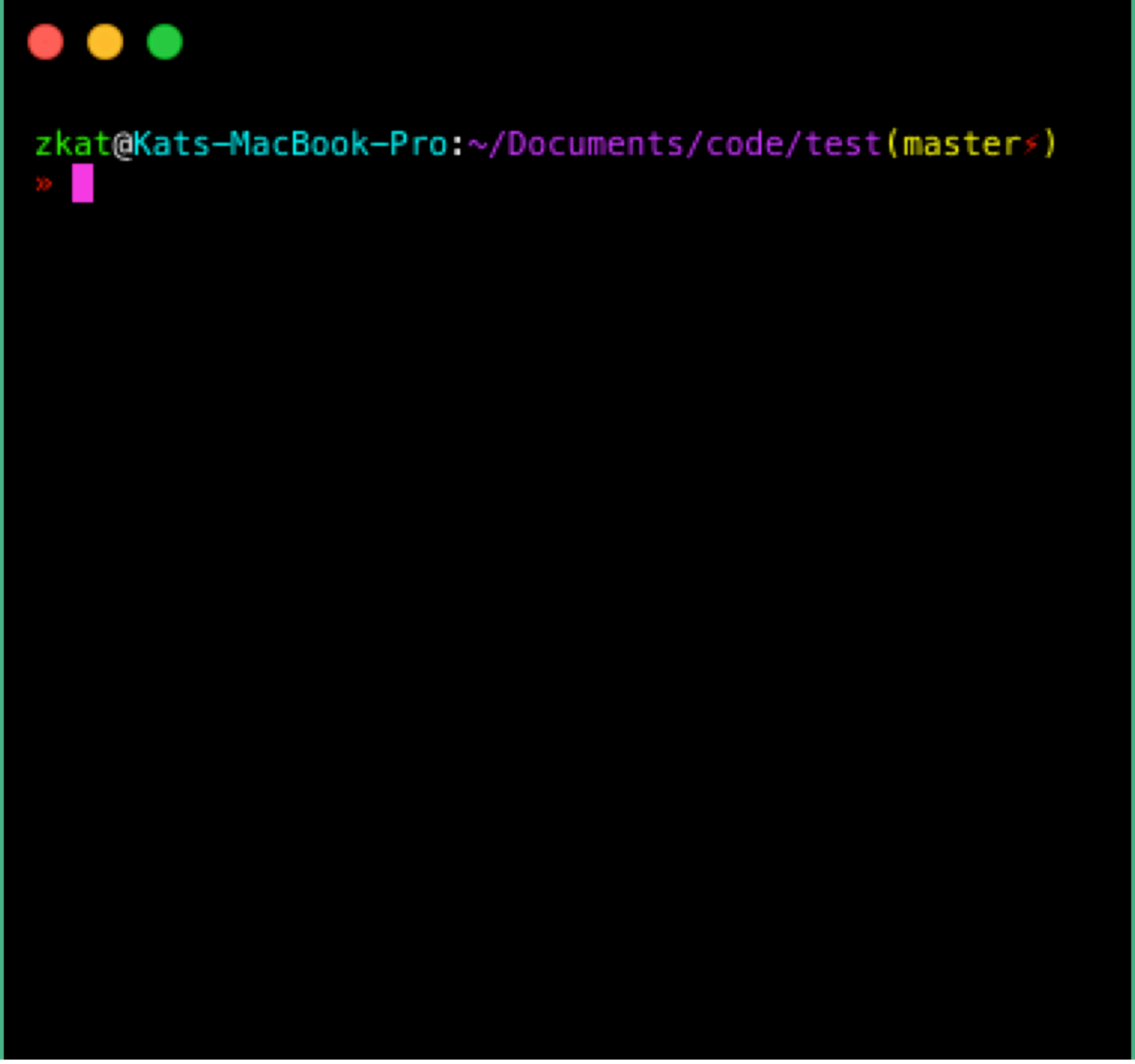


```
zkat@Kats-MacBook-Pro:~/Documents/code/test(master⚡)
 »
```

```
[<-happen  5: reattach-to-user-namespace  (6: test)]
```

AUTOMATIC CORRUPTION RECOVERY

This one's pretty huge. This is not something any existing JS package manager can pull off. Yarn, for example, can't detect whether its *cache contents* are corrupted. So if a file goes missing or there's some disk error? You're outta luck. npm5, though, will seamlessly redownload bad data. It will *never* give you corrupted data.



```
zkat@Kats-MacBook-Pro:~/Documents/code/test(master⚡)
» 
```

OTHER FEATURES

» shasum in npm-shrinkwrap.json (+ fast cache fetches)

» semver support for git: zkat/pacote#semver:^7

» sha512 support (security! important!)

» streamed extraction (Unity needs this!)

There's a few other cool things available already:

THE FUTURE: NPM@5.1 - NPM@6

- Isaac's node-tar@3 makes extract 30% faster
- Parallel execution of node-gyp scripts
- New node_modules tree reader
- package.json/npm-shrinkwrap.json hashing
- Improved packument fetch phase (it's structurally throttled atm)
- shrinkwrap and --save by default
- local cache mirrors

As I said, npm@5 isn't the end of the story. We've got several more features and optimizations coming right after release:

THE FUTURE IS EVEN FASTER

THANX

Questions? Concerns? Incoherent Screaming?

GET IT NOW!

```
npm i -g npm5 \
  --registry=https://registry.internal.npmjs.com
```