

TINK

GESTIONADOR DE JAVASCRIPT PRÓXIMA-GENERACIÓN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Hola, mi nombre es Katerina Marchán. También me conocen como @maybekatz en el internet, y so la mantenedora y arquitecta del npm CLI.

Pero no estoy aquí hoy para hablar de npm en sí. Estoy aquí para contarles sobre un experimentito que he estado haciendo con un proyecto llamado Tink, que espero que establezca un nuevo precedente para los gestionadores de paquetes de JavaScript en el futuro.

Time: TKTK

¿QUÉ TIENE QUE PASAR?

NPM COMO CREADOR DE `node_modules/`

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Pero antes que empiece con el "qué", quiero hablarles un poquito sobre el "por qué". Más al punto, quiero hablar sobre lo que npm necesita hacer para que las aplicaciones de JavaScript funcionen. Y en fin, sólo hay un quehacer con npm: poner un bonche de cosas en tu `node_modules/` local, lo más rápido posible, para que entonces las aplicaciones puedan consumir archivos de ahí.

Pues bien, eso me parece bastante directo por su cuenta, pero qué involucra lograrlo?

Time: TKTK

EL PROCESO DE INSTALACIÓN

O SEA: ¿POR QUÉ COÑO TOMA 10 MINUTOS HACER ESTA MIERDA?

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Para compartir un poco de contexto, quiero hablar sobre el problema en general, y que pasos hemos tomado en el camino para reducir los problemas que encontramos. Ahora, el proceso que voy a describir se lo comparten mas o menos todos los gestionadores de JavaScript que conozco. Todos tienen optimizaciones diferentes, y truquitos para manejar cada paso, pero todos tienen que hacer más o menos lo mismo en algún punto u otro para set compatibles. Vamos a ver...

Time: TKTK

EL PROCESO DE INSTALACIÓN

1. LEE DEPENDENCIAS LOCALES (SI EXISTEN)
2. GESTIONA METADATOS DE PAQUETES AUSENTES
3. CALCULA ÁRBOL + ACCIONES (APLANANDO DESDE NPM@3)
4. DESCARGA + EXTRAER PAQUETES QUE FALTEN
5. EJECUTA SCRIPTS DE INSTALACIÓN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Esto es un resumen de lo que un gestor de JavaScript tiene que hacer. Todos nosotros hacemos todo esto en algún momento, o tenemos nuestras propias optimizaciones para cada paso, pero los requisitos son los mismos. Vamos a ver cada uno paso a paso.

Time: TKTK

EL PROCESO DE INSTALACIÓN

1. LEE DEPENDENCIAS LOCALES (SI EXISTEN)
2. GESTIONA METADATOS DE PAQUETES AUSENTES
3. CALCULA ÁRBOL + ACCIONES (APLANANDO DESDE NPM@3)
4. DESCARGA + EXTRAE PAQUETES QUE FALTEN
5. EJECUTA SCRIPTS DE INSTALACIÓN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

El primer paso es este -- leer las dependencias locales, si tienes algunas. Estos días probablemente sólo notes este paso si haces una instalación después de haber instalado normalmente. Es decir, cuando acabas de ejecutar npm y lo haces otra vez. Eso normalmente no toma más de un par de segundos, pero definitivamente se nota. Yarn tiene una optimización interesante con esto donde escribe un archivo de metadatos dentro de `node_modules/` basado en el hash de yarn . Lock, y si los dos son iguales, Yarn simplemente dice "que se joda" y confía que `node_modules/` está bien.

npm ha sido un poco terco con esto, porque no consideramos que esto es un caso que haya que optimizar, y si hacemos lo mismo que Yarn, no podemos reparar tu `node_modules/` automáticamente. En nuestra opinión, esto vale la pena versus ahorrarse un par de segundos. No sé, tal vez cambiemos de opinión porque esos segundos le importan bastante a algunas personas.

En fin, este paso se nota, pero en verdad no toma mucho de tu tiempo.

Time: TKTK

EL PROCESO DE INSTALACIÓN

1. LEE DEPENDENCIAS LOCALES (SI EXISTEN)
2. GESTIONA METADATOS DE PAQUETES AUSENTES
3. CALCULA ÁRBOL + ACCIONES (APLANANDO DESDE NPM@3)
4. DESCARGA + EXTRAER PAQUETES QUE FALTEN
5. EJECUTA SCRIPTS DE INSTALACIÓN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

El cuento es un poco diferente con esta, por lo menos por un tiempo. Resulta que hacer literalmente miles de pedidos para calcular nuestro árbol es una operación bastante intensiva con el network. Y en los tiempos antes de npm@5, este paso era de los que más tiempo tomaban. Entonces qué paso? Bueno, básicamente, los lockfiles pasaron. Lockfiles son super convenientes para programadores, claro, pero en verdad son un cache del árbol calculado por este paso, y por eso es que nos gusta tanto. Una vez hayas instalado to proyecto, nadie en tu equipo tiene que repetir este paso hasta que añadan o remuevan una de las dependencias. Esta también es la razón porque le digo a todo el mundo que use package-lock.json con sus librerías, no solamente con sus aplicaciones. La verdad que mejoran la vida.

Time: TKTK

EL PROCESO DE INSTALACIÓN

1. LEE DEPENDENCIAS LOCALES (SI EXISTEN)
2. GESTIONA METADATOS DE PAQUETES AUSENTES
3. CALCULA ÁRBOL + ACCIONES (APLANANDO DESDE NPM@3)
4. DESCARGA + EXTRAER PAQUETES QUE FALTEN
5. EJECUTA SCRIPTS DE INSTALACIÓN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Este próximo paso nunca ha sido super lento por su cuenta, pero quería mencionarlo porque sí nos importa que sea por lo menos lo suficiente rápido, y quiero que ustedes sepan que existe.

Básicamente, esto es lo que calcula que dependencias tienen que ser movidas, removidas, añadidas, copiadas, o lo que sea. Desde npm@3, nosotros también calculamos la versión plana de tu node_modules, y ahora es bastante rápido el proceso.

Bueno, siguiendo a delante...

Time: TKTK

EL PROCESO DE INSTALACIÓN

1. LEE DEPENDENCIAS LOCALES (SI EXISTEN)
2. GESTIONA METADATOS DE PAQUETES AUSENTES
3. CALCULA ÁRBOL + ACCIONES (APLANANDO DESDE NPM@3)
4. DESCARGA + EXTRAER PAQUETES QUE FALTEN
5. EJECUTA SCRIPTS DE INSTALACIÓN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Este paso. Este puto paso. Este es el paso más lento y difícil de optimizar de todos, principalmente porque se limita por todos los recursos que normalmente hacen que las cosas tomen tiempo. Vamos a ver este paso un poco más en profundo...

Time: TKTK

EL PASO DE EXTRACCIÓN

- > DESCARGA (NETWORK)
- > PARSING + UNZIP (CPU)
- > ESCRIBIENDO A DISCO (DISCO, ALMACENAJE)

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

TKTK translate?

En fin, tenemos ciento de megabytes que tienen que ser transferidos sobre el network, y una vez tengamos todos los datos, tenemos que usar un montón de recursos de CPU para analizar y hacer unzip a los paquetes. Y cuando terminemos con eso, hay que hacer un montón más de I/O de disco.

Así que los tres mayores sospechosos que causan problemas de rendimiento están presentes de un modo u otro, y todo depende de las pautas de uso del usuario.

Lo que quiero decir aquí es que este paso es donde todo lo malo y lo lento ocurre, y este paso ha sido el blanco de optimizaciones para básicamente todos los gestionadores de paquetes del mundo de JavaScript. Y me imagino que es el mismo cuento con gestionadores de otros idiomas también.

Time: TKTK

LO QUE **HEMOS** HECHO

- > CACHEAR AGRESIVAMENTE
- > CACHEAR **DESPUÉS** DE EXTRAER
- > HACER **"HARD LINK"** DE UN CACHE **CENTRAL**
- > REUSAR CACHE CON **PLUGINS**

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Pues qué podemos hacer sobre esto? Bueno, por ejemplo, podemos cachear los paquetes en sí agresivamente, y diferentes estrategias de cacheo tienen diferentes pros y contras. npm, por ejemplo, cachea los paquetes en sí, una estrategia que ahorra espacio de disco, pero quiere decir que tenemos que analizar y extraer los paquetes por completo cada vez que los instalamos.

Yarn los almacena después de la extracción, así que usa más espacio pero los deja hacer copias de los archivos rápidamente, en vez de tener que extraer los paquetes.

pnpm hace algo muy interesante aquí. Ellos usan un cache extraído, igual que Yarn, pero en vez de copiar, hacen "hard links" de todos los archivos cuando instalan. Es un poco sorprendente, pero esto no es mucho más rápido que hacer copias, pero lo que sí logra es ahorrarse espacio de disco, porque sólo pagas el costo de cada archivo una vez, a través de todo tu sistema.

Y como parece que no me puedo callar sobre Yarn, ellos también tienen una característica muy interesante llamada PnP que parcialmente trata con los problemas de este paso. Ellos usan un plugin, y todos los sistemas que quieren usar PnP tienen que ser modificados, pero la ventaja es que ni siquiera tienen que hacer copias: todos los archivos se leen directamente del cache.

Time: TKTK

EL PROCESO DE INSTALACIÓN

1. LEE DEPENDENCIAS LOCALES (SI EXISTEN)
2. GESTIONA METADATOS DE PAQUETES AUSENTES
3. CALCULA ÁRBOL + ACCIONES (APLANANDO DESDE NPM@3)
4. DESCARGA + EXTRAER PAQUETES QUE FALTEN
5. EJECUTA SCRIPTS DE INSTALACIÓN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Así que estamos haciendo mucho trabajo para hacer este paso en particular un poco mejor. Y, sabes, no hay sorpresa, porque es el paso que más cuesta. Voy a volver a esto pronto, y qué más podemos hacer, pero por ahora vamos a acabar con esta explicación.

Así que terminamos toda nuestra descarga y extracción, y entonces...

Time: TKTk

EL PROCESO DE INSTALACIÓN

1. LEE DEPENDENCIAS LOCALES (SI EXISTEN)
2. GESTIONA METADATOS DE PAQUETES AUSENTES
3. CALCULA ÁRBOL + ACCIONES (APLANANDO DESDE NPM@3)
4. DESCARGA + EXTRAE PAQUETES QUE FALTEN
5. EJECUTA SCRIPTS DE INSTALACIÓN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

...tenemos que analizar todos los paquetes que acabamos de instalar y ejecutar cualquier run-script de instalación. Este paso no es necesariamente el más grande, porque normalmente sólo hay un par de run-scripts, y la mayoría no son muy lentos, pero esto todavía es un paso que se puede notar en algunos proyectos.

Ahora, npm no hace esto, pero algunos gestionadores de paquetes paralelizan este paso. Es un poco riesgoso y complicado, pero si puede hacer que cuando tienes -múltiples- scripts de instalación, las cosas pueden ir mucho más rápido, por ejemplo si usas oniguruma y node-sass juntos.

En mi opinión, la verdad es que es mejor usar algo como node-pre-gyp en esos paquetes, pero eso no siempre es posible...

Pero entonces, después de todo esto y todos estos pasos, lo importante que hay que reconocer es que...

Time: TKTK

node_modules/ ES MASIVO

Y DONDE LOS SUEÑOS SE VAN PARA MORIRSE

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

La verdad es que `node_modules`, aunque es una buena abstracción y nos ha servido bastante bien, también ha causado muchísimos dolores de cabeza entre la usabilidad y el tiempo perdido y el espacio de disco abusado.

Ryan Dahl dió una charla el año pasado sobre Node y Deno, y la verdad que mucho de lo que dijo tiene mucha razón.

Time: TKTK

CON TODO Y ESO

- > DEPENDENCIAS DE PROYECTO AISLADAS SON BUENAS.
- > NO TENER 'DEPENDENCY HELL' ES BUENO.
- > YA TENEMOS UN ECOSISTEMA DE 1MM DE PAQUETES.

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Pero que podemos hacer con todo esto? Es fácil quejarse y hablar de cuan mala es una idea, pero en fin yo creo que lo bueno sale adelante de lo malo.

(run through the list)

Time: TKTK

HAGAMOS LO QUE PODAMOS CON ESTO? 🧚

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Y la verdad que es nuestra responsabilidad, tanto la de los programadores de gestionadores como la de la comunidad en general, hacer lo que podemos con todo esto. Y yo, obviamente, tengo mis propias ideas sobre esto, y les di una pequeña pista de lo que es...

Time: TKTK

TINK

GESTIONAMIENTO DE PAQUETES

EN LA PLATAFORMA

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Lo que yo creo que deberíamos hacer ahora es mover el gestionamiento de paquetes en sí directamente a la plataforma, en vez de a una utilidad externa. Y eso es lo que hace tink. Pero, qué quiere decir esto, y que podemos hacer con esto, una vez lo tengamos?

Time: TKTK

\$ tink sh
EN VEZ DE
\$ node

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Esto es lo que estoy tratando de decir. Con tink, ya no usas node directamente. Usas esta herramienta llamada `tink sh` que envuelve a node en sí con par de parches al módulo `fs` que hace que la magia funcione.

Pero qué magia?...

Time: TKTK

node_modules/ **VIRTUAL**


- > **REMUEVE** node_modules/ FÍSICO
- > ARCHIVOS **DE-DUPLICADOS GLOBALMENTE** POR **HASH**
- > DESCARGA DE DEPENDENCIAS **AUTOMÁTICA**
 - > NO. MÁS. **npm install**

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

La idea aquí es que si controlamos la plataforma, podemos controlar lo que pasa cuando alguien trata de leer directamente de node_modules. Una vez tengamos esa habilidad al nivel de plataforma, se habren las compuertas a un sinnúmero de posibilidades.

Time: TKTK

node_modules/ **VIRTUAL**

- > **REMUEVE** node_modules/ FÍSICO 
- > ARCHIVOS **DE-DUPLICADOS GLOBALMENTE POR HASH**
- > DESCARGA DE DEPENDENCIAS **AUTOMATICA**
 - > NO. MÁS. **npm install**

TINK: A NEXT-GEN PM | KATERINA MARCHÁN |  @MAYBEKATZ

Quiere decir que nos podemos deshacer de todas estas copias de node_modules/ en el sistema sin cambiar el cargador de módulos de node, o cambiar las APIs que esperan los paquetes. En lo que le respecta a los paquetes, están accedendo el sistema de archivos de la misma manera que normalmente hacen. Y esto quiere decir que es compatible con pequeños detalles como `__dirname`, como `fs.readFile` para leer archivos dentro del paquete, y todas esas cosas.

Time: TKTk

node_modules/ **VIRTUAL**

- > **REMUEVE** node_modules/ FÍSICO
- > ARCHIVOS **DE-DUPLICADOS GLOBALMENTE POR HASH** 📌
 - > DESCARGA DE DEPENDENCIAS **AUTOMÁTICA**
 - > NO. MÁS. **npm install**

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦 @MAYBEKATZ

Pero si no están dentro de node_modules/, donde están todos los archivos? Bueno, en vez de copiarlos o hacer "hard links", los mantenemos todos en un cache global, igual que pnpm, pero de-duplicados al nivel de hash. Eso quiere decir que si tienes 5 versiones del mismo paquete, sólo vas a almacenar copias de los archivos que cambiaron entre cada versión, no de los paquetes enteros. Almacenar usando los hashes también ayudan a que el acceso sea **muy** rápido.

Time: TKTK.

node_modules/ **VIRTUAL**

- > **REMUEVE** node_modules/ FÍSICO
- > ARCHIVOS **DE-DUPLICADOS GLOBALMENTE POR HASH**
- > DESCARGA DE DEPENDENCIAS **AUTOMÁTICA** 👉
 - > NO. MÁS. **npm install**

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦 @MAYBEKATZ

Pero mira, podemos hacer más. Como controlamos la plataforma, por qué no vamos y descargamos dependencias por tí, sólo cuando las necesitas? Por qué no saltamos el paso de descargar cosas que nunca vas a usar, como los READMEs y CHANGELOGs?

Tink puede bloquear cuando tratas de leer algo del cache local y fallas, y, automáticamente, descargar lo necesario. Esto hace que node funcione más como un navegador de internet, un browser. No sé cómo le dicen aquí.

Time: TKTK

node_modules/ **VIRTUAL**

- > **REMUEVE** node_modules/ FÍSICO
- > ARCHIVOS **DE-DUPLICADOS GLOBALMENTE** POR **HASH**
- > DESCARGA DE DEPENDENCIAS **AUTOMÁTICA**
- > 😎 **NO. MÁS.** **npm install** 😎

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦 @MAYBEKATZ

Y claro, todo esto quiere decir que no tienes que hacer `npm install` nunca más. Simplemente haces `tink add` y `tink rm` para añadir y remover dependencias, y no tienes que preocuparte por nada más ni por esperar por un paso de instalación. Todo lo hace tink automáticamente, cuando lo necesites.

Time: TKTK

¿PERO NO ES INSEGURO PARCHEAR FS?

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Bueno, pues tal vez se sienten un poco preocupados por todo esto. Parchear fs en sí? No es riesgoso?

Time: TKTK

¿PERO NO ES INSEGURO PARCHEAR FS?

¡ELECTRON HACE LA MISMA COSA!
¡Y FUNCIONA!

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Pero a mí no me preocupa. Esto es lo mismo que hace Electron y les ha funcionado bastante bien. También podemos aprender de sus lecciones así que yo creo que esto va a funcionar bien al final.

Time: TKTK

PERO HAY MÁS

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

OTRAS COSAS QUE TINK FACILITA

- > SOPORTE DE TYPESCRIPT, ESM Y JSX
- > CHECKSUM DE ARCHIVOS DE DEPENDENCIAS
 - > CERO CONFIGURACIÓN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Hay aún otras cosas que tink nos deja hacer: tink tiene soporte de typescript, módulos ecma-script, y JSX incorporado.

También ejecuta un checksum de seguridad muy barato cada vez que lees un archivo del cache global, así que puedes confiar que lo que sacas del sistema es lo que se supone que sea.

Y finalmente, mi parte favorita -- todo esto involucra *cero* configuración de tu parte, y no tienes que instalar nada más que tink en sí.

Time: TKTK

CERO CONFIGURACIÓN

(NINGUNA, NADA)

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Y quiero enfatizar esto un poco más. Todo esto es gratis, sin ningún loader de webpack o plugin de Jest. Después que use las APIs de Node, todo funcionará igual.

Time: TKTK

UNA GIRA DE LA HERRAMIENTA

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Vamos a darnos una gira a la herramienta, para que tengan una idea más completa de lo que estoy hablando...

Time: 5s

YO QUIERO...

EJECUTAR MI APP DE NODE

`$ tink sh [file.js]`

`0: $ tish [file.js]`

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Esta es la característica principal de la herramienta. Casi todo sobre lo que he hablado pasa por este comando en sí. Recuerden que no tienen que hacer `npm install` ya. Ese paso se fué. Todo lo que tienen que hacer es ejecutar `tink sh` y todas tus dependencias se descargan y extraen cuando se necesiten por tu app. Esto también funciona como un shell interactivo!

Yo sé que yo he hecho mucho esfuerzo para mejorar el rendimiento de npm y herramientas relacionadas, pero lo más importante para mí, en fin, es reducir el volumen de trabajo que tengan que hacer ustedes para lograr lo que quieren. Quiero que tengan que instalar lo menos posible, aprender lo menos posible, y que pueden enfocarse en su propia productividad.

La habilidad de simplemente ejecutar tu aplicación y dejar que la plataforma se encargue de todo es super importante para mí para lograr el formato de trabajo que creo que ustedes merecen en su día a día. Lo mejor que yo puedo hacer es asegurarme que su gestor de paquetes desaparezca y que no sea algo de lo que se tengan que preocupar jamás.

Time: TKTK

YO QUIERO...

EJECUTAR UN BINARIO/DEVDEP LOCAL

```
$ tink exec <cmd>
```

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Ok pues cuantos de ustedes conocen sobre npx aquí? Levanten las manos.

Bueno, pues `tink exec` es más o menos como `npx`. Para el resto de ustedes, `npx` es una herramienta que está incluida con `npm` en sí, y una de las cosas que hace es que te deja ejecutar binarios locales. Por ejemplos, si instalas `jest` como devDependency en vez de global, puedes usar `npx jest` para invocar a `jest`, sin tener que instalarlo globalmente. Ni siquiera tienes que configurar un `run-script` para ello. Ejecutar tus binarios locales usando `tink` quiere decir que podemos usar la misma lógica para pre-instalar cualquier archivo necesario para tu binario, que `tink shell` usa para tus scripts.

Time: TKTK

YO QUIERO...

INSTALAR DEPS **ANTES** DE EJECUTAR

\$ tink prepare

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Pero y si no quieres que se ponga un poco más lenta tu app con todo esto de gestionamiento automático? Para eso, puedes usar `tink prepare`. Este comando calienta el cache para que tengas todos los archivos necesarios, para que todos tus run-scripts y binarios estén listos antes de que uses `tink sh`. Funciona igual o más de rápido que si hubieras usado `npm install` antes. Se puede decir que este comando es más o menos el reemplazo de `npm install`, excepto que no escribe casi nada a `node_modules`, así que va a ser mucho más rápido.

Lo importante aquí es que este comando es completamente opcional, y muchas veces, tal vez no va a ahorrarte tiempo en absoluto, porque no importa si lo hagas antes o después, esto se tiene que hacer.

Time: TKTK

YO QUIERO...

TENER `node_modules/` DE VERDAD
`$ tink unwind`

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Pero y si quiero un `node_modules/` físico de todas maneras?

En ese caso, hay este comando llamado `tink unwind` que puedes usar para extraer completamente tus dependencias a `node_modules`. Con esto puedes usar tu editor para parchear algo, usar herramientas de build que no sean JavaScript, y básicamente lo que quieras de la manera normal. Esto es como `tink prepare`, pero con el paso extra en que extraes los paquetes por completo.

Time: TKTK

YO QUIERO...

HACER **DEBUG** A UNA DEPENDENCIA

```
$ tink unwind <dep>
```

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Y si quiero hacer esto con una sola dependencia, en vez de instalar todo?

Bueno, en ese caso usas `tink unwind <dep>`, y sólo va a instalar a esa dependencia, más sus dependencias, en `node_modules`. Eso te deja parchear lo que quieras, hacer debug, etcetera.

Y esto es algo que quiero enfatizar -- por la manera que funciona tink, cualquier cosa dentro del `node_modules` físico tiene prioridad sobre la versión virtual.

Además, si tratas de hacer algo como `fs.writeFile` para escribir a `node_modules`, tink va a escribir el archivo directamente a tu `node_modules`, que ayuda muchísimo con la compatibilidad.

Este comando también se hace automáticamente al nivel de paquetes individuales si tienes una dependencia con scripts de instalación, para mejorar la compatibilidad aún más, por ejemplo con `node-gyp` u otros sistemas sobre los que no tenemos control que necesitan archivos físicos.

Time: TKTK

YO QUIERO...

AÑADIR/REMOVER PAQUETES

- > \$ tink add <pkg>
- > \$ tink rm <pkg>
- > \$ tink update <pkg>

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Entonces si lo más cerca a `npm install` solamente maneja dependencias existentes, cómo añadimos y removemos dependencias, sin tener que editar `package.json`?

Pues con el trio clásico: `add`, `rm`, y `update`.

Estos comandos hacen básicamente lo que dicen en inglés: les das los nombres de paquetes, y entonces te escriben un `package.json` y `package-lock.json` nuevo.

Y por si acaso, los tres comandos se vuelven interactivos si no les pasas ningún argumento. Eso quiere decir que vas a poder buscar dependencias interactivamente y seleccionarlás de un menú. Lo mismo para removerlas y ponerlas al día.

Time: TKTk

YO QUIERO...

HACER TYPECHECK, PRUEBAS, LINTER

\$ tink check

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

El último comando que les voy a mostrar es una utilidad llamada `tink check`. Lo más chulo de esto es que es un solo paso para toda tu verificación y pruebas. Este comando, por ejemplo, va a chequear tu typescript, sin tener que instalar typescript como dependencia. Si decides instalar Typescript, va a usar esa versión para verificar. Pero no se siente super nice poder hacer esto sin tener que instalar y manejar a typescript? Hacerle lint? Es super nice.

Time: TKTK

TODO JUNTO...

- › tink sh [script.js]
- › tink exec <comando>
- › tink prepare
- › tink unwind [pkg]
- › tink {add, rm, update} [pkg...]
- › tink check

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

En resumen, estos son los comandos principales que forman parte de tink. Es algo un poco nuevo que aprender, pero este es el gestor de paquetes que me gustaría que npm fuera, y espero que sea el gestor que ustedes usen en el futuro. Les gusta? :)

¿Y AHORA QUÉ?

- > **TERMINAR CON EL PROTOTIPO**
- > **ESTABLECER UN EQUIPO BASADO EN RFCS**
- > **ESCRIBIR PARTES EN RUST/WASM PARA VELOCIDAD**
- > **TRABAJAR CON NODE CORE PARA INTEGRARLO**

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Así que de aquí a donde vamos? Bueno, lo que es tink ahora es solamente un prototipo, porque todo esto fue hecho como prueba de concepto. Pero yo creo que esa fase ya ha terminado y es hora de implementar el gestor de verdad.

El próximo paso es emocionante. Vamos a crear un equipo abierto, basado en comité y RFCs que incluye más contribuidores externos de los que el CLI de npm tradicionalmente ha tenido. Esto nos va a ayudar mucho a terminar este proyecto tan grande, y además asegurarnos que la comunidad pueda poner de su parte y que lo que se produzca sea lo que la comunidad en verdad quiere. El proceso de RFCs va a ayudar con esto, y quiere decir que **tú** puedes ayudar a crear esto. Y si tienen problemas con el inglés a mí no me importa, pueden producir documentos en español y yo se los traduzco y me comunico con ustedes en español. Lo que sea que necesiten para que no sea solamente un bonche de gringos en la bahía produciendo los productos que los afectan tanto a ustedes.

Una cosa que quiero mencionar es que tengo la intención de escribir partes del gestor usando Rust y wasm, que ahora está disponible en todas las versiones activas LTS de Node, así que por fin lo podemos usar! Si te interesa aprender Rust, o aprender como usarlo con wasm, esto es un proyecto donde puedes hacer todo eso.

Finalmente, una vez el gestor esté escrito y madure un poco, la intención es integrarlo con Node como el nuevo gestor por defecto, con varias adiciones para que sea más compatible con lo que ahora es npm. No sé que se va a necesitar para esa negociación, y sé que es un salto grande tener un gestor tan diferente que haga cosas como parchear fs, pero estoy segura que lo podremos lograr.

Time: TKTK

¿Y AHORA QUÉ?

- > TERMINAR CON EL PROTOTIPO
- > ESTABLECER UN EQUIPO BASADO EN RFCS
- > ESCRIBIR PARTES EN RUST/WASM PARA VELOCIDAD
- > TRABAJAR CON NODE CORE PARA INTEGRARLO
- > Y UNA COSITA MÁS...

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Ah, claro... y una cosita más...

¿UN NUEVO REGISTRO?

GITHUB.COM/ENTROPIC-DEV/ENTROPIC

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Ah, si, probablemente algunos de ustedes se enteraron hace par de semanas de un nuevo registro llamado Entropic, hecho por un grupo de ex-empleados de npm.

Cuántos de ustedes ya oyeron sobre esto? (manos)

Bueno, para el resto de ustedes... hay un grupo de miembros de la comunidad que están escribiendo un registro nuevo, y libre de npm, Inc, que está hecho para que se pueda federar, para que todo el mundo pueda establecer un ecosistema de multiples servidores, gratis. Lo que esperamos es poder poner este nuevo registro en las manos de la fundación OpenJS, en ves de las manos de una corporación, para el beneficio de la comunidad.

Y eso suena bueno y todo, pero lo importante aquí es que tink va a admitir la API nueva de este registro, que es diferente de la corriente que usa registry.npmjs.org. Y lo bueno de la nueva API es que abre puertas a unas características muy interesantes de tink...

Time: TKTK

¿UN NUEVO REGISTRO?

- > DESCARGA ARCHIVOS INDIVIDUALES EN VEZ DE PAQUETES
- > HASTA 40% REDUCCIÓN DE DESCARGA/ALMACENAJE
- > `tink sh` BUSCA SÓLO ARCHIVOS REQUERIDOS
- > MEJOR PUBLICACIÓN DE MONOREPOS

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Para darles una idea de por qué importa tanto el registro nuevo para algo como tink...

La API en sí es bastante pequeña: es un par de endpoints nuevos que nos dejan enumerar los archivos que forman parte de un paquete, y descargar esos archivos individualmente, usando el hash. Eso es todo, esa es básicamente la idea para el nuevo API.

Y qué ganamos con eso? Bueno, por ejemplo, estimamos que puede reducir el tamaño de tus descargas hasta 40%. Eso quiere decir que las instalaciones van a ir mucho más rápido. Eso quiere decir que si tienes una conexión más lenta que fibra, vas a tener que esperar mucho menos para que termine el proceso y te puedas poner a trabajar. Eso quiere decir que no vas a descargar ningún más README, CHANGELOG, or directorio de `test/` al menos que tu código los use.

Esta reducción se hace posible porque `tink sh` descarga paquetes sólomente cuando los necesita, así que sólo descargas la primera vez que tratas de leer un archivo, y entonces lo cacheamos agresivamente. Esta es una de mis razones favoritas para trabajar en tink.

Finalmente, y esto es interesante: Usar algo así abre la posibilidad de cambiar por completo la forma en que trabajamos con librerías de monorepos, por ejemplo, lodash y babel.

Time: TKTK

MEJOR PUBLICACIÓN DE MONOREPOS

- > 147 PAQUETES EN MONOREPO DE BABEL
 - > TODOS PUBLICADOS CON LERNA
- > PAQUETES INSTALADOS REQUIEREN VERSIONES IGUALES

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Como probablemente ya saben, estas librerías involucran un montón de sub-herramientas, y generan docenas o cientos de paquetes que publican bajo sus scopes, para que la gente no tenga que depender del paquete entero.

Esto puede causar muchos problemas. Por ejemplo, los usuarios pueden tener problemas sincronizando todas las diferentes versiones, si dependen de múltiples paquetes, algo que es completamente común. Del lado del publicador, esto es un setup súper frágil porque causa que el proceso de publicación tenga tendencia a tener errores, y entonces necesitan más y más herramientas para que, por ejemplo, puedan resumir el proceso de publicación. Mas que esto causa problemas si alguien instala un paquete nuevo antes que el resto estén publicados. Así que todos estos paquetes pequeñitos tienen que ser publicados a la misma vez, usando la misma versión, y entonces tienes que lidiar con el registro de npm que es eventualmente consistente, así que no todas las versiones van a estar visibles, y todo termina en caos.

Time: TKTk

MEJOR PUBLICACIÓN DE MONOREPOS

> PUBLICA babel COMO UN PAQUETE

> `import '@babel/foo' -> import 'babel/foo'`

> SOLAMENTE SUBPAQUETES USADOS SE DESCARGAN

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Bueno, con tink y entropic, esto ya sería necesario! Lodash y babel podrían publicar un solo paquete omnibus que incluye todos los subpaquetes, pero cuando los instalas con tink, solamente los archivos de esos paquetes que usas son los que se van a descargar. Todo el mundo gana, y solamente tienes que tener una línea en `to package.json` con `lodash` o `babel`, en vez de docenas de diferentes paquetes.

Bueno, eso es todo. Como les dije, esto está en proceso ahora mismo como parte del proyecto Entropic, pero creo que va a cambiar mucho el futuro de como la gente trabaja con JavaScript. Que lo disfruten!

Time: TKTk

EN RESUMEN

- TINK IS EMOCIONANTE! Y UN TRABAJO EN ELABORACION
 - NODE_MODULES/ VIRTUALES
 - TYPESCRIPT, ESM, JSX POR DEFECTO
- ENTROPIC COMO NUEVO REGISTRO DE JAVASCRIPT

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ

Time: 25s

AYUDA A CREAR A TINK

¡ACOMPÁÑANOS!

- > [GITHUB.COM/ENTROPIC-DEV/ENTROPIC](https://github.com/entropic-dev/entropic)
 - > [GITHUB.COM/NPM/TINK](https://github.com/npm/tink)
 - > [@MAYBEKATZ](https://twitter.com/maybekatz) EN TWITTER
- > [@ZKAT](https://github.com/zkat) EN GITHUB -- ¡PATROCÍNAME!

TINK: A NEXT-GEN PM | KATERINA MARCHÁN | 🐦@MAYBEKATZ