UNIDAD 1 – Automatización de Tareas

1. [UNIDAD 1: Introducción](https://learn.nextu.com/mod/page/view.php?id=3587&pid=P_WEB_APPS)

Bienvenido a **Herramientas, Build Systems y tecnologías emergentes**

La automatización de tareas permite ejecutar aquellas que, por lo general, están escritas como plugins. En esta Unidad aprenderás cómo integrar a tus proyectos las más usadas en el desarrollo de JavaScript, siempre considerando cuáles son las más beneficiosas. Es decir, reconociendo cuáles tareas facilitarán el trabajo o, por el contrario, cuáles lo hacen más complejo.

Puntos de aprendizaje

Unidad 1: Automatización de tareas

Los objetivos del Programa que corresponden a esta Unidad son:

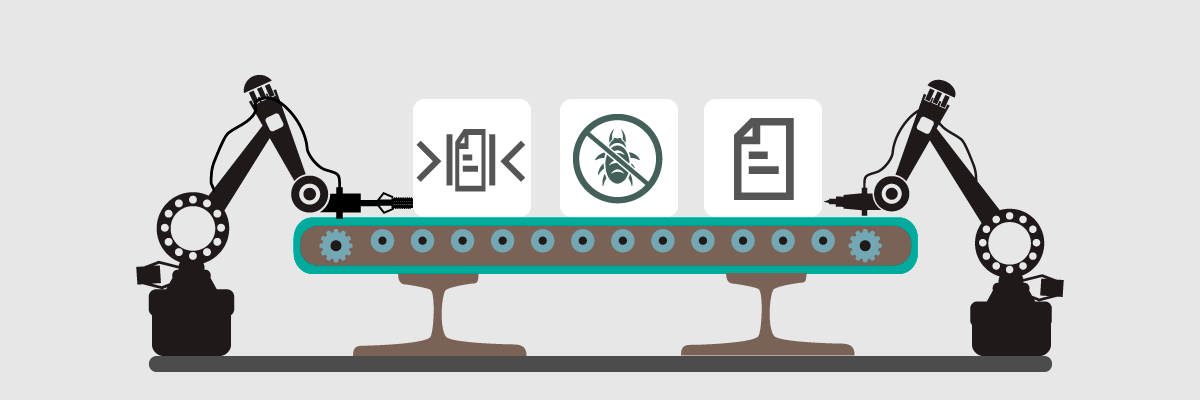
* Crear ambientes de programación y a gestionar recursos.
* Implementar la construcción de sitios web a partir de componentes.

Los objetivos del Curso 8 que corresponden a esta Unidad son:

* Identificar tendencias web.
* Aplicar el concepto ambiente web.
* Usar herramientas para la gestión de tareas.

1. Lección 1: Task Runners (Automatización de tareas)
   1. [Lectura: Introducción a la automatización de tarea](https://learn.nextu.com/mod/lesson/view.php?id=3588&pid=P_WEB_APPS)

**La Automatización de Tareas**



¿Qué es?

La automatización nos permite ejecutar tareas que por lo general están escritas como plugins. Veamos cuáles podemos ejecutar y cómo llamarlas.   
  
Las siguientes son algunas de las tareas más usadas en el desarrollo en Javascript. Úsalas como inspiración para integrarlas a los proyectos en los que creas que puedan ser beneficiosas pero recuerda: siempre debes pensar si implementar una nueva tarea facilitará tu flujo de trabajo o, por el contrario, lo complica. ¿Por qué? A veces podemos tener infraestructuras complejas de tareas que en lugar de reducirnos el trabajo lo incrementan.

Tareas más comunes de automatización

1. Minify

Consiste en comprimir código, ya sea js - css - html o cualquier otro archivo, en un archivo de lógica, estilo y estructura equivalente que pesa menos, eliminando espacios innecesarios, reduciendo los nombres de variables a pocas letras y eliminando comentarios.

2. Linting

Se refiere a mostrar posibles bugs del código sin ejecutarlo y advertencias o errores que se pueden agregar, para ayudarnos a seguir guías de estilo o evitar patrones de código que indican malas prácticas.

3. Transpiling

Es el proceso de convertir un lenguaje interpretado a otro, es equivalente a compilar.   
En este caso el lenguaje objetivo será Javascript pero podemos partir desde otras versiones de Javascript (ES6), código en otros lenguajes ya existentes (Java, Scala, Ruby, Python, C/C++) o lenguajes creados exclusivamente para ser transpilados a js (Coffeescript o Typescript).

4. Licensing and Versioning

Podemos aprovechar cada vez que construimos nuestro proyecto para escribir en el compilado final (build) la licencia de nuestro código, información de autoría y versiones.

5. Sourcemaps

Si hacemos procesos de minificacion o transpiling podemos generar mapas de código que nos permitan ver cómo se transformó nuestro código y utilizar esta información para debugear el código.

6. Rename, delete, create y map functions through files

También es muy fácil ejecutar tareas propias de los sistemas de archivos, reemplazar, cortar pedazos, renombrar archivos, borrarlos, crear nuevos archivos copiando anteriores o procesar archivos con funciones y luego guardarlos nuevamente con otros nombres. Todas estas tareas son sencillas y muy poderosas.

7. Documentación

Existen múltiples herramientas que nos permiten crear la documentación en el momento de compilación para tenerla siempre al día.   
Usos más avanzados nos permiten, por ejemplo, generar la documentación y, si estamos desplegando a producción, actualizar el sitio estático que mantiene la documentación con la nueva versión de la documentación y, además, medir cómo es el cubrimiento de la documentación de la base de código, a medida que crece.

8. Ejecución y cubrimiento de pruebas

Así como podemos generar la documentación cada vez que construimos el proyecto, también podemos correr las pruebas unitarias que hayamos creado y ver el cubrimiento, así como podemos detener la carga del código a la producción si éstas fallan y emitir alarmas a los desarrolladores.

9. Despliegue automático de integración continua

La integración continua es sencilla de implementar si utilizamos algunas tareas que ejecuten las herramientas respectivas. La rama ‘production’ de tu repositorio puede ser probada (tests unitarios) y luego desplegada así: si todo sale bien, todos los días en la madrugada, justo cuando el tráfico en el sitio es el mínimo, y si algo sale mal, puedes configurar emails o alertas automáticas a los desarrolladores.

10. Uso automático de repositorios

Imagina configurar útiles metodologías de trabajo como por ejemplo: que cada vez que tu código compile y estén todas las pruebas bien, automáticamente se actualice tu rama de desarrollo de git, o con un sólo comando combine la rama de producción, active hooks de git en los servidores relevantes y despliegue el código.

11. Servir aplicaciones

Existen también múltiples herramientas que fácilmente nos dan servidores de desarrollo con características como sockets, o que incluso pueden emular bases de datos de mockup o ambientes particulares.

12. Descargar URL's

Podemos escuchar URLs específicas donde se desplieguen los recursos y descargarlos para ponerlos en nuestra versión actual de la aplicación.

13. Watchers

Nos permiten observar continuamente archivos y actuar ante el cambio en estos. Entonces, si guardamos el archivo en el editor o un archivo en particular, por ejemplo, podemos aprovechar para hacer un git pull - merge - push y servir de nuevo la versión de desarrollo.

14. Livereload

Existen plugins que también nos permiten empujar nuevas versiones de la aplicación, a clientes que ya están ejecutándose, usando sockets de manera automática.

15. Procesamiento de imágenes

Muchas tareas como descargar, generar thumbnails, cortar o procesar imágenes para añadir marcas de agua o efectos, pueden ser ejecutadas en tiempo de compilación de las tareas.

16. Servir y subir archivos vía FTP / SSH

También podemos crear servidores FTP directamente desde nuestras tareas o pedir y enviar archivos a un servidor vía SSH, lo que nos permite desplegar nuestra aplicación directamente en nuestras tareas.

17. Renderizar componentes web en el servidor

Podemos utilizar emulación del browser en el lado del servidor para que si nuestros clientes no pueden ejecutar el Javascript de nuestras páginas les sirvamos una versión ya renderizada en html. Estas versiones son generadas por las tareas con herramientas como phantomjs.

18. Construir múltiples versiones de nuestras aplicaciones

Si nuestro código de Javascript, por ejemplo, alimenta un sitio web, una aplicación de electrón (escritorio) y una aplicación móvil, podemos construirlos todos a la vez , probarlos todos a la vez, o hacer cualquier tarea que se nos ocurra con tantos objetivos de compilación como queramos.

19. Enviar emails

Crear y enviar emails de manera automatizada para avisar de cambios de versión, despliegues de código, alarmas, entre otros.

20. Gestionar archivos estáticos en CDN’s

Si nuestra infraestructura de archivos estáticos se vuelve muy compleja, podemos administrar múltiples CDN (Redes de envío de contenido), cambiar estas URLs en nuestros archivos estáticos y automatizar toda esta infraestructura de contenido fácilmente.  
Con la variedad de tareas que podemos automatizar, podremos realizar diversos proyectos muy interesantes, llevando tareas manuales muy complejas a una forma simple y fácil, complementando nuestros proyectos con muy buenas tecnologías.

* 1. Actividad 1

Tiempo de pensar

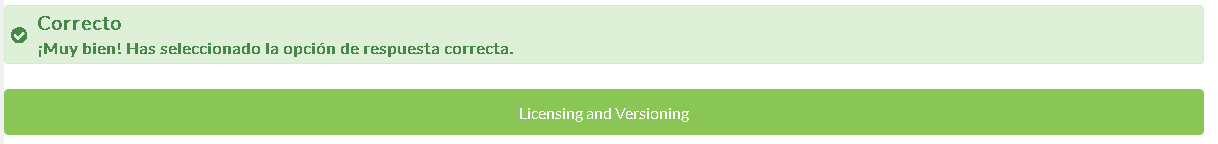
Ahora que sabes que la automatización nos permite ejecutar tareas que por lo general están escritas como plugins y cuáles son las más usadas, te invitamos a realizar esta actividad.   
  
Instrucciones:  
  
Asocia los nombres de algunas de las tareas más comunes de automatización, ubicados en la columna izquierda, con su respectiva definición, en la columna derecha.  
¡Adelante!



* 1. Actividad 2

Tiempo de pensar

Recuerda que siempre debes pensar si implementar una nueva tarea facilitará tu flujo de trabajo o, por el contrario, lo complicará. ¿Por qué? A veces podemos tener infraestructuras complejas de tareas que en lugar de reducirnos el trabajo lo incrementan.  
  
Instrucciones:  
  
A continuación te presentamos una definición.  
Deberás leerla y determinar a cuál de las tareas de automatización que se plantean en las opciones de respuesta corresponde.¡Ánimo!  
  
Difinición:  
  
Con esta tarea podemos aprovechar cada vez que construimos nuestro proyecto para escribir, en el compilado final (build), la licencia de nuestro código, información de autoría y versiones.



* 1. Ejecutando mi primera tarea

Comando para ejecutar un prueba con npm:

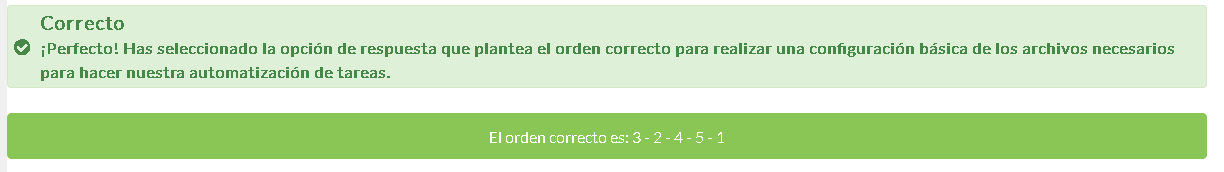
npm run test

* 1. Actividad 3

Tiempo de pensar

Ahora que sabes que la automatización te permite ejecutar tareas que por lo general están escritas como plugins, te invitamos a realizar la siguiente actividad.  
  
Instrucciones:  
  
A continuación presentaremos unos enunciados que corresponden a los pasos para realizar una configuración básica de los archivos necesarios para realizar nuestra automatización de tareas. El objetivo es que realices un análisis que te permita ordenarlos mentalmente y que, luego, elijas la opción de respuesta que plantea el orden correcto. Entonces, lee con atención y, cuando estés seguro, responde.  
  
Enunciados:

1. Para ejecutar las tareas que definimos en este lugar, realizamos la siguiente operación en la consola “npm run test”. Esta operación imprime el mensaje escapeado dentro de comillas.
2. Después aparecerá dentro de la consola una solicitud de ciertos campos que responderemos con los datos solicitados.
3. Abrimos un terminal de nuestra computadora y nos dirigimos con nuestra terminal al lugar de hayamos creado nuestro proyecto. Por medio del siguiente comando iniciamos un nuevo proyecto nodejs npm init.
4. Confirmamos y hemos creado nuestro nuevo proyecto. Aparece un nuevo archivo package.json
5. Debido a que este archivo está en formato json, buscamos la clave nombrada con scripts. Allí encontramos la clave con el nombre de test que viene por defecto cuando creamos nuestro primer proyecto node.



2.6 Laboratorio

LABORATORIO DE  
**Instalación y Configuración**

La empresa de conciertos Luz Azul cuenta con un sitio web estático donde ponen la información relevante a los instrumentos que utilizan y, en el futuro, pondrán la información de los conciertos realizados. El director de la orquesta quiere empezar a promocionar los conciertos de manera internacional y ha encomendado al director de tecnología que se implemente una estrategia de internacionalización para la página. También continuamente el equipo de fotografía e imagen cambia los recursos que se muestran en la página web y esto implica esfuerzos por parte del equipo de tecnología de la empresa.

El director de tecnología te ha pedido que, como desarrollador web, crees una estrategia de internacionalización en la cual se produzca el sitio web de Luz Azul en cada uno de los idiomas que se planean introducir. Además ha resaltado otros aspectos: debes crear una manera en la que los recursos sean definidos por el equipo de fotografía e imagen y se integren al sitio web de manera automática.  
En una revisión de código del sitio actual se encontró que usa foundation, jquery y what-input.js como dependencias de código javascript / css. Debes instalar estas dependencias en el sitio y la infraestructura de tareas que diseñamos.

Contenido:

Para el desarrollo de este laboratorio debes descargar el código base que contiene la página estática de Conciertos Luz Azul.

Contenido

Instrucciones

* 1. **Estructura del proyecto:**  
       
     Descarga el código base e inspecciónalo.  
     Indica la estructura del proyecto y cuál es la función de cada carpeta. Luego describe la estructura del archivo index.html, indicando donde se incluyen las hojas de estilo, los scripts de Javascript y las imágenes que se muestran en la página.  
       
     El proyecto consiste de un archivo index.html donde está el código html del proyecto. Adicionalmente tenemos 3 carpetas css, js y img.
     + **Carpeta js:** Contiene los archivos Javascript del proyecto, adicionalmente contiene una carpeta vendor con las dependencias en Javascript.
     + **Carpeta img:** Se encuentran todas las imágenes que provee el equipo gráfico
     + **Carpeta css:** Contiene las dependencias de css de la librería Foundation y el css de la aplicación en app.css
  2. **Manejo de scripts:**  
       
     Diseña las tareas que se deben ejecutar para los manejos de scripts de las librerías usadas.
     + Los scripts deben ser descargados por un **manejador de paquetes** que permite, a su vez, descargar las dependencias de manera local.
     + Se debe mantener un archivo para saber qué librerías y en qué versión deben estar instaladas. **(Package.json)**
     + Se debe hacer una **tarea** que revise qué archivos son dependencias de la aplicación y los contenga en una carpeta haciendo **copias de los archivos** para la versión que se va a construir.
     + Estos archivos .js pueden juntarse en un solo archivo **vendor-bundle.js,** por ejemplo, utilizando una **tarea que los concatene.**
     + Los scripts de la aplicación también pueden juntarse con **vendor-bundle.js** en **app.js** para servir un sólo archivo Javascript con otra **tarea de concatenación.**
     + El nombre del archivo a importar debe ser inyectado en el **index.html** por una tarea de **reemplazamiento.**
  3. **Manejo de recursos:**  
     Diseña las tareas que implementan una infraestructura para el manejo de los recursos gráficos, creados por el equipo de fotografía.  
     Para eso, realiza una **tarea de reemplazamiento,** tomando las llaves desde el archivo json y las reemplaza por las correctas en todos los html.
  4. **Estrategia de localización del sitio web:**  
     Diseña la infraestructura de tareas requerida para generar las versiones localizadas del sitio web. Debes tener archivos html esqueleto que sean los que se tomarán de base, utilizando un diccionario de etiquetas reemplazar en cada bundle para cada sitio web estático generado. Index.html finalmente detectara el lenguaje a cargar y luego inyecta el html necesario según el lenguaje.
     + Carpeta con plantillas html
     + Carpeta con diccionarios en JSON
     + Construcción del bundle html para cada lenguaje con tareas de reemplazamiento y luego copia.
     + Meta Plantilla de index.html que reemplaza el index.html y mediante javascript detecta qué lenguaje debe ser cargado para luego, cargarlo.

Buenas Prácticas

Recuerda que puedes guardar varias versiones de archivos intermedios a medida que tu cadena de procesamiento de tareas se va completando. Usualmente es mejor servir al final la menor cantidad de archivos posibles para reducir la cantidad peticiones.

1. Lección 2: Gulp.JS
2. Lección 3: Grunt.JS
3. Lección 4: NPM como herramienta para automatización de tareas
4. UNIDAD 1: Prueba