Herramientas de Línea de Comando para Desarrolladores

De
Piotr Gaczkowski
toptal.com
15 min
Ver original

***Nota**: se ha reducido el texto eliminando algunas partes para hacer la lectura un poco más corta.*

¿Alguna vez has visto alguna película con hackers? La mayoría de las veces, estas personas se muestran trabajando frente a monitores que muestran algún tipo de terminal (generalmente con un fondo oscuro y un primer plano claro). Esta terminal, a su vez, tiende a estar inundada de caracteres que aparentemente tienen algún significado para la persona que los lee.

Tales representaciones de hackers en acción a menudo son burladas por desarrolladores profesionales, e incluso hay algunos programas que simulan varios efectos de "hackers", solo por diversión.

Sin embargo, en el mundo real, las herramientas de línea de comandos no se utilizan por su valor de entretenimiento.

Por qué todavía utilizamos las herramientas de interfaz de línea de comandos

Este artículo se centra en el aspecto práctico del uso de herramientas de interfaz de línea de comandos (CLI). Conocer los comandos de la CLI y usar herramientas de calidad puede hacerlo más productivo y también pueden abrir las puertas a varios enfoques de la automatización que son mucho más prácticos con las interfaces textuales que con las aplicaciones GUI.

Puede mejorar al realizar tareas repetitivas en GUI, hasta el punto de que sus múltiples clics se escuchen como uno solo. El problema es que esto no mejorará la eficiencia de un script especializado. Además, realizar las mismas operaciones de forma manual introduce tanto una carga cognitiva adicional como una mayor posibilidad de error humano. Como de costumbre, dependemos de computadoras para manejar tareas que los humanos pueden considerar aburridas, repetitivas o abrumadoras.

Vale la pena saber que una herramienta de terminal puede ofrecer varios tipos de interfaces. Hay elementos no interactivos como ls, que simplemente toman los parámetros y proporcionan la salida. Existen interfaces interactivas o semi interactivas que se encuentran con mayor frecuencia en los gestores de paquetes. ("¿Está seguro de que desea continuar con la instalación desde un origen no verificado?") Luego, hay interfaces de usuario textuales (TUI), que son aplicaciones de GUI interactivas diseñadas para ajustarse a las limitaciones de una terminal. Probablemente el más famoso es Midnight Commander (mc), un clon extremadamente popular (de los años 90) Norton Commander.

Herramientas esenciales de la línea de comandos

Si deseas convertirse en un habitante de la consola, debe equiparse con un conjunto mínimo de herramientas de desarrollo de línea de comandos: lo esencial. Las cosas con las que definitivamente no puede vivir son una **shell interactiva** (apunta a algo moderno con una conveniente terminación de pestañas) y un **editor de texto**.

Shell

Lo primero que verá al abrir una terminal es un shell. Esta es la parte que hace posible la interacción entre el usuario y la máquina. Interpreta tus comandos, los divide en nombres de programas y argumentos, y ejecuta todos los comandos de shell que le arrojas.

Históricamente, ha habido muchos tipos diferentes de shell. Entre los más populares se encuentran **csh** (C Shell) y varias implementaciones de Bourne Shell (generalmente conocido simplemente como **sh**). Bourne Shell se extendió a Korn Shell, que también ganó algo de tracción y todavía está siendo utilizado por sus entusiastas. Csh es actualmente el shell predeterminado en algunos sistemas BSD, mientras que casi todos los demás sistemas operativos tipo UNIX prefieren algún tipo de Bourne Shell. Las distribuciones de Linux tienden a favorecer bash mientras que Mac OS X viene con zsh como la opción predeterminada.

Existen otras posibilidades, pero son mucho menos populares, excepto Microsoft PowerShell en sistemas Windows. PowerShell está inspirado en parte por los shells interactivos de UNIX como zsh y en parte por .NET runtime. En lugar de tratar todo como texto, un concepto común en el mundo UNIX, permite la manipulación de datos orientada a objetos.

Aunque Microsoft PowerShell es bastante popular en el ámbito de Windows, muchos programas con orígenes de UNIX (los más notables son Git, Autotools o Make) tienden a preferir alguna variación de Bourne Shell. Debido a esto, han nacido proyectos como msys (incluido con Git para Windows), Cygwin, o el reciente [WSL] de Microsoft. Si deseas una sensación similar a Linux en Windows, MSys es la mejor opción aquí. Si desea un entorno Linux con todas las funciones capaz de ejecutar binarios estándar de Linux, entonces WSL es el camino a seguir. Para algo intermedio: API de UNIX, pero compilada como un ejecutable de Windows (solo úsala cuando realmente sepas por qué la necesitas) -Cygwin es la respuesta.

Editor

Una vez que te familiarices con tu shell, querrás adquirir algunas habilidades útiles. Como la mayoría del trabajo de codificación gira en torno a la escritura de texto (código, README, mensajes de confirmación), un buen conocimiento de los editores de texto interactivos es esencial. Hay muchos para elegir, y dado que un editor es una de las herramientas más necesarias para cualquier desarrollador, probablemente haya tantas opiniones sobre cuál editor es el mejor.

La mayoría editores de texto populares pueden separarse en dos grupos básicos: **Editores de texto simple** y **editores de texto programables**.

Ambos pueden ser excelentes para escribir código, pero, como su nombre lo indica, los programables ofrecen la capacidad de dar forma y personalizar el editor para que se ajuste perfectamente a sus necesidades. Sin embargo, esto tiene un precio, ya que también tienden a tener una curva de aprendizaje más pronunciada y pueden requerir más tiempo para establecerse.

Editores de texto básicos

Entre los editores de texto simple, GNU Nano es el más extendido. En realidad, es un clon del editor pico, por lo que si uno no está disponible en su sistema, puede probar el otro. Otra alternativa más moderna para ambos es el editor micro. Si quieres algo simple y extensible al mismo tiempo, este es un buen lugar para comenzar.

Editores de texto programables

Muchos desarrolladores dependen de editores programables de diferentes campos, como Vim y GNU Emacs. Ambos editores pueden ejecutarse en la consola o en modo GUI, y ambos tuvieron un impacto en los enlaces de teclas encontrados en otro software. Ambos ofrecen no solo una API sino también lenguajes de programación reales incorporados. Emacs se centra en LISP y Vim usa su propio VimL, pero también ofrece interfaces con otros lenguajes de scripting populares (como Lua, Perl, Python o Ruby). También vale la pena mencionar un enfoque más reciente de Vim, llamado Neovim, ya que está empezando a recibir seguidores serios.

Puede ser algo confuso, pero también hay un editor llamado vi que es un predecesor de Vim (que, dicho sea de paso, significa "**Vi** mejorado"). Es mucho más simple que Vim, pero si tienes suficiente confianza para escribir en Vim, no debería ser un desafío para ti si te resulta necesario usar vi.

Dado que pico/GNU Nano y vi/Vim generalmente están preinstalados en varios sistemas, es una buena idea al menos comprender sus conceptos básicos (salir de Vim es un problema notoriamente difícil para los principiantes). De esta manera, si necesita editar algo en una máquina remota, estará listo independientemente de qué editor ya esté allí. En su dispositivo privado, puede usar cualquier editor que le resulte más cómodo.

Administrador de paquetes

En este punto, puede comenzar a pensar en instalar todo el software antes mencionado en su máquina. Un problema es que cada una de las herramientas tiene instrucciones de instalación diferentes. A veces, necesitas descargar fuentes y compilarlas tú mismo, a veces obtienes el binario autónomo, y algunas veces obtienes lo que se llama **paquete binario**, que generalmente significa un ejecutable comprimido junto con algunos metadatos.

Para facilitar el proceso de instalación de software, los creadores de sistemas operativos llegaron con el concepto de administradores de paquetes. En pocas palabras, un administrador de paquetes es como una tienda de aplicaciones para CLI y aplicaciones de escritorio. Precede a las tiendas de aplicaciones reales por algunas décadas. El problema es que casi todos los sistemas tienen su propio administrador de paquetes. Debian, Ubuntu y las distribuciones derivadas de GNU / Linux usan APT, las distribuciones basadas en Red Hat prefieren yum o DNF, otras distribuciones de Linux tienen medios más exóticos para instalar software y también lo hacen los diferentes clones de BSD. Además de los administradores de paquetes integrados, también hay instalados por el usuario como Chocolatey para MS Windows y Homebrew para Mac OS X/Mac OS. Cuando desee escribir instrucciones sobre cómo instalar su programa, puede terminar escribiendo casos para cada uno de esos sistemas. Parece demasiado, ¿no?

Afortunadamente, el último de los sistemas mencionados, Homebrew, puede ser el más portátil, gracias a Linuxbrew, un puerto de Homebrew para sistemas GNU / Linux. Lo curioso es que

incluso funciona en WSL si quieres tener una experiencia de usuario similar en Microsoft Windows. Ten en cuenta que WSL no es oficialmente compatible.

Entonces, además de la portabilidad, ¿qué más puede ofrecer Homebrew? En primer lugar, no interfiere con los paquetes del sistema, por lo que todo lo que instala reside en una capa separada del sistema operativo. Además, no se necesitan permisos de raíz para instalar paquetes. Por lo tanto, puedes tener paquetes de sistema que sean estables y probados, pero al mismo tiempo verifica tus versiones más nuevas sin sacrificar la estabilidad del sistema.

Si querías probar los editores, mencioné anteriormente que todo lo que necesita hacer en un sistema con Homebrew o Linuxbrew es ejecutar este comando:

`brew install emacs micro nano vim neovim`.

La Cosa Brillante

Lo que ya hemos discutido es indudablemente útil para el trabajo. Pero también hay aplicaciones que, aunque no son necesarias, aún brindan comodidad a la vida cotidiana. Puede que no los necesites, pero siempre vale la pena conocerlos.

Git UI

Es probable que al menos uno de los proyectos en los que trabaja usa Git como un sistema de control de versiones. Si bien es completamente poderoso, la CLI de Git no es el pináculo de la excelente experiencia del usuario. Para ahorrarte un poco de estrés leyendo todas las opciones en la ayuda de Git `$ SUBCOMMAND`, te recomiendo que revises tig. Ofrece una agradable interfaz de usuario de consola para las operaciones que se benefician de ella, como `log` o `blame`.

Otra herramienta que tiene como objetivo ayudar a los usuarios de GIt es fac, que es un acrónimo de *Fix All Conflicts*. Como habrás adivinado, es útil cuando te encuentras con conflictos mientras realizas fusiones o rebases. Es una alternativa a otras herramientas de combinación como vimdiff.

Línea de comando en la práctica

Entonces, ¿qué tiene de atractivo la línea de comandos que compensa las horas dedicadas a aprender a usar el shell, el editor y todos los conmutadores de varias aplicaciones? La respuesta corta es **productividad**, que proviene de dos cosas:

- Una es que cuando se te presenta solo una ventana de terminal y nada más, puedes concentrarte más intensamente, ya que no hay mucho que te distraiga. No aparecen notificaciones, ni anuncios, ni fotos de bonitos gatitos. Solo tú y tu objetivo.
- Lo segundo es la automatización. Puedes poner varias acciones combinadas con frecuencia en un script y llamarlo más tarde como un todo en lugar de escribirlas todas a mano cada vez. Puedes volver rápidamente a un comando particularmente complejo que alguna vez escribió al buscar en el historial de su shell. Básicamente, puedes grabar y reproducir cualquier cosa, y el código está disponible como una documentación de lo que hiciste.

La capacidad de agregar alias también contribuye a las ganancias. Por ejemplo, a menudo me encuentro creando commits en Git actualizando el mismo hasta que sea perfecto (por el momento).

Una vez que preparo los archivos deseados, ejecuto `git carmh`. No intente buscarlo en el manual, ya que es mi alias privado que significa `commit --amend --reuse-message = HEAD`. Ahorra algo de tipeo de seguro.

La cosa es que la gente se aburre repitiendo las mismas acciones una y otra vez, y el aburrimiento reduce el enfoque. Esto puede conducir a errores y errores. La única forma de evitarlos es no entrelazar las acciones de alto enfoque y bajo enfoque. El código de escritura es de alto enfoque y la revisión de un mensaje de compromiso y el contenido es de alto enfoque, pero cuando necesita repetir varios clics mecánicos aquí y allá para llegar a la etapa de revisión de compromiso, es probable que su enfoque se reduzca. La línea de comando no está, por supuesto, libre de tales actividades mecánicas, pero gracias a la automatización, puede evitar la mayoría de ellas.

Otras exploraciones

Hay programas de línea de comando mucho más interesantes que existen, y si estás interesado en ellos, le recomiendo consultar la lista curada de Awesome Shell de algunos de los mejores comandos de la shell disponibles hoy

La mayoría de las aplicaciones GUI tienen su contraparte terminal. Eso incluye navegadores web, clientes de correo electrónico, clientes de chat (IRC, Slack, XMPP), suites PIM u hojas de cálculo. Si conoces algún buen programa que no he mencionado, bríndalos en comentarios.