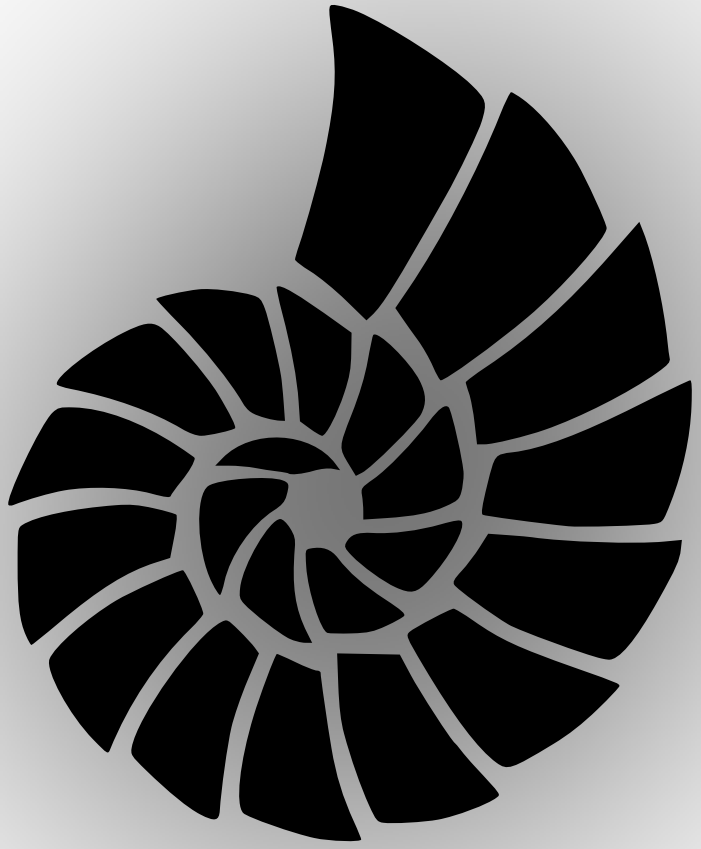
- El arte de la línea de comandos



Traducido por

Lyx
Drymer
Maxxcan
Zorro
Fanta
Suggie
y Frangor
de El Binario

Índice general

0.1. El arte de la línea de comandos

La soltura del uso de la consola es una destreza a menudo abandonada y considerada arcaica, pero mejorará tu flexibilidad y productividad como si fueras un i ngeniero de una forma obvia y sutil. Esta es una selección de notas y consejos de como usar la línea de comandos de consola que encontré útil cuando trabajaba en Linux. Algunos consejos son básicos, y otros bastante específicos, sofiscitados, u .ºscuros". Esta página no es larga, pero si usas y recuerdas todos los puntos, sabrás lo suficiente.

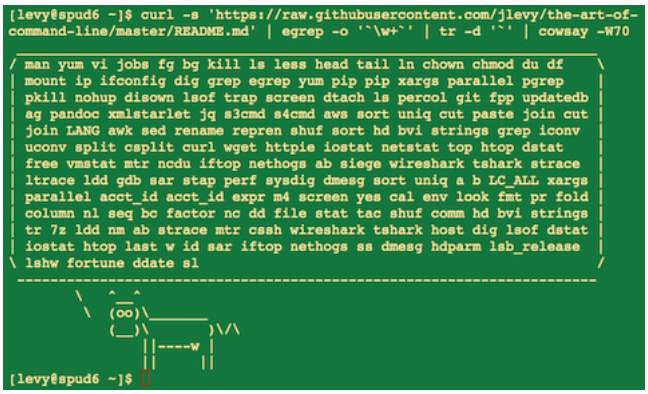


Figura 1: curl -s 'https://raw.githubusercontent.com/jlevy/the-art-of-command-line/master/README.md'q | egrep -o '\w+' | tr -d "' | cowsay -W50

0.1.1. Meta

Objetivo:

- Esta guía es tanto para el principiante como para el experimentado. Los objetivos son *amplitud* (todo importa), *especificidad* (dar ejemplos concretos del uso más común), y *brevedad* (evitar lo que no sea esencial o que se puedan encontrar fácilmente en otro lugar). Cada consejo es esencial en alguna situación o ahorra tiempo significativamente en comparación con las alternativas.
- Esta escrito para Linux, con excepción de la sección "Sólo para MacOS(#macos-only)". Muchos de los otros puntos aplican o pueden ser instalados en otros sistemas UNIX o MacOS (o incluso Cygwin).
- Se hace especial hincapié en Bash interactivo. Aunque muchos de los consejos se pueden aplicar para otras consolas y a la programación general en Bash.
- Incluye tanto comandos "estándar" Unix como aquellos que requieran la instalación de paquetes – siempre que sean lo suficientemente importantes como para incluirlos.

Notas:

- Para mantener el texto en una página, el contenido está expuesto como referencia. Se asume que el lector es suficientemente inteligente como para investigar en detalle una vez que se tenga la idea general. Se utiliza `apt-get`/ `yum`/ `emerge`/ `dnf` /`pacman`/ `pip`/ `brew` (según proceda) para instalar los nuevos programas.
- Usar http://explainshell.com/Explainshell para obtener información detallada sobre usos y funciones de comandos, opciones, tuberías, etc.O

0.1.2. Fundamentos

- Aprenda los conocimiento básicos sobre Bash. De hecho, escriba `man bash` y al menos échele un vistazo a todo; es bastante fácil de seguir y no es tan largo. Alternar entre shells puede ser agradable, pero Bash es poderoso y siempre está disponible (aprender *solo* zsh, fish, etc., aunque resulte tentador usarlo en tu propio portátil, le restringe en muchas situaciones, tales como el uso de servidores).

- Aprenda a usar correctamente al menos un editor de texto. Idealmente Vim (`vi`), ya que realmente no tiene competencia para la edción aleatoria en una terminal (incluso si usted usa Emacs, un gran IDE, o un editor moderno, hipster y alternativo la mayor parte del tiempo).

- Conozca como leer la documentation con `man` (para los curiosos, `man man` muestra las secciones enumeradas, ej. 1 es para comandos "regulares", 5 es para archivos/convenciones, y 8 es para administración). Busca páginas de man con `apropos`. Sepa que alguno de los comandos no son ejecutables, son órdenes embebidas en Bash, y que puede acceder a la ayuda con `help` y `help -d`.

- Aprenda sobre redirección de salida y entrada $>$ y $<$ y tuberias utilizando $|$. Sepa que $>$ sobreescribe el archivo de salida y $>>$ añade. Aprenda que es el stdout y el stderr.

- Aprenda sobre expansión de archivos glob con $\star$ (y tal vez `?` y $\{\ldots\}$) y las citas y la diferencia entre los apóstrofes doble `"` y simple `'`. (Para ver más sobre expansión de variable mire más abajo.)

- Familiarizate con la administración de trabajos en Bash: `&`, **ctrl-z**, **ctrl-c**, `jobs`, `fg`, `bg`, `kill`, etc.

- Conoce `ssh`, y lo básico sobre autenticación sin contraseña, via `ssh-agent`, `ssh-add`, etc.

- Adminisración de archivos básica: `ls` y `ls -l` (en particular, aprenda el significado de cada columna que aparece en `ls -l`), `less`, `head`, `tail` y `tail -f` (o incluso mejor, `less +F`), `ln` y `ln -s` (aprenda las diferencias y ventajas entre los enlaces duros y los blandos), `chown`, `chmod`, `du` (para un rápido resumen del uso del disco: `du -hk *`). Para administración del sistema de archivos, `df`, `mount`, `fdisk`, `mkfs`, `lsblk`.

- Administración de redes básico: `ip` o `ifconfig`, `dig`.

- Conozca bien las expresiones regulares, y varias etiquetas (flags) para `grep`/`egrep`. El `-i`, `-o`, `-A`, y `-B` son opciones dignas de ser conocidas.

- Aprenda a usar de `apt-get`, `yum`, `dnf` o `pacman` (dependiendo de la distribución (distro)) para buscar e instalar paquetes. Y asegurate de que tienes `pip` para instalar herramientas de linea de comando basadas en Python (unas cuantas de las que vienen más abajo son más fáciles de instalar vía `pip`).

0.1.3. De uso diario

- En Bash, se usa **Tab** para completar los argumentos y **ctrl-r** para buscar, a través, del historial de comandos.
- En Bash, se usa **ctrl-w** para borrar la última palabra, y **ctrl-u** para borrar todo hasta el inicio de la línea. Se usa **alt-b** y **alt-f** para moverse entre letras, **ctrl-k** para eliminar hasta el final de la línea, **ctrl-l** para limpiar la pantalla. Ver `man readline` para todos los atajos de teclado por defecto en Bash. Hay una gran cantidad de ellos. Por ejemplo **alt-.** cambia, a través, de los comandos previos, y **alt-**** expande los comandos.
- Alternativamente, si tu amas los atajos de teclado al estilo de vi, usa `set -o vi`.
- Para ver los últimos comandos, `history`. También existen abreviaciones, tales como, `!$` (último argumento) y `!!` último comando, aunque sin fácilmente remplazados con **ctrl-r** y **alt-.**.
- Para volver al directorio de trabajo previo: `cd -`
- Si estas a mitad de camino de la escritura de un comando pero cambias de opinión, presiona **alt-#** para agregar un `#` al principio y lo agregas como comentario (o usar **ctrl-a**, **#**, **enter**). Para que puedas entonces regresar a éste luego con el comando *history*.
- Se usa `xargs` (o `parallel`). Este es muy poderoso. Nota que tu puedes controlar muchos ítems ejecutados por línea (`-L`) al igual que parallelism (`-P`). Si tu no estás seguro si esto lo hace correctamente, usa `xargs echo` primero. También `-l` es útil. Como ejemplo:

```
bash
find -name '*.py* | xargs grep alguna_funcion
cat hosts | xargs -l{} ssh root@hostname
```

- `pstree -p` es útil para mostrar el árbol de procesos.
- Se usa `pgrep` y `pkill` para encontrar o señalar procesos por su nombre (`-f` es de mucha ayuda).
- Conocer varias señales que puedes enviar a los procesos. Por ejemplo, para suspender un proceso, usa `kill -STOP [pid]`. Para obtener la lista completa consulta `man 7 signal`
- Usa `nohup` o `disown` si quieres mantener un proceso de fondo corriendo para siempre.
- Verifica que procesos están escuchando vía `netstat -lntp` o `ss -plat` (para TCP; agrega `-u` para UDP).
- Consulta también `lsof` para abrir sockets y archivos.
- Usar `alias` para crear atajos para comandos comúnmente usados. Por ejemplo, `alias ll="las -latr"` crea un nuevo alias `ll`

- En Bash scripts, usa `set -x` para depurar la salida. Utiliza el modo estricto cuando se posible. Utiliza `set -e` para abortar en errores. Utiliza `set -o pipefail` también, para ser estrictos sobre los errores (aunque este tema es un poco delicado). Para scripts más complejos, también se puede utilizar `trap`.

- En Bash scripts, subshells (escritos con paréntesis) son maneras convenientes para agrupar los comandos. Un ejemplo común es para moverse temporalmente hacia un directorio diferente de trabajo, Ej:

[] do something in current dir (cd /some/other/dir other-command) continue in original dir

- En Bash, considere que hay muchas formas de expansión de variables. Verificar la existencia de una variable: `${name:?error message}`. Por ejemplo, si un script Bash requiere un único argumento, solo escriba `input_file=${1:?usage: $0 input_file}`. Expansión aritmética: `i=$(( (i + 1) % 5 ))`. Secuencias: `{1..10}`. Reducción de strings: `${var%suffix}` y `${var#prefix}`. Por ejemplo:

  Si `var=foo.pdf`, entonces `echo ${var%.pdf}.txt` imprime `foo.txt`.

- La salida de un comando puede ser tratado como un archivo, via `<(commando)`. Por ejemplo, compare local `/etc/hosts` con uno remoto:

```
diff /etc/hosts <(ssh somehost cat /etc/hosts)
```

- Conocer acerca "here documents" en Bash, así también `cat <<EOF ...`.

- En Bash, redireccionar ambas salida estándar y error estándar, vía: `comando > logfile 2>&1`. Frecuentemente, para garantizar que un comando no haya dejado abierto un archivo para controlar la entrada estándar, vinculado al terminal en el que te encuentras, esta también como buena practica puedes agregar `</dev/null`.

- Usa `man ascii` para una buena tabla ASCII, con valores hexadecimal y decimales. Para información de codificación general, `man unicode`, `man utf-8`, y `man latin1` son de utilidad.

- Usa `screen` o https://tmux.github.io/`tmux` para multiplexar la pantalla, especialmente útil en sesiones remotas y para desconectar y reconectar a una sesión. Una alternativa más minimalista para persistencia de la sesión solo sería `dtach`.

- En ssh, conocer como hacer una conexion tunelada con `-L` o `-D` (y de vez en cuando `-R`) es útil, Ej. para acceder sitio web desde un servidor remoto.

- Esto puede ser útil para hacer algunas optimizaciones para su configuración ssh; por ejemplo, `~/.ssh/config` contiene la configuración para evitar desconexiones en ciertos entornos de red, usar comprensión (la cual es muy útil con **scp** sobre conexiones con un ancho de banda pequeño), y multiplexar canales para el mismo servidor con un archivo de control local:

```
TCPKeepAlive=yes
ServerAliveInterval=15
ServerAliveCountMax=6
Compression=yes
ControlMaster auto
ControlPath /tmp/%r@%h:%p
ControlPersist yes
```

- Unas pocas opciones más relevantes para ssh son relativas a la seguridad y deben ser activadas con cuidado, Ej. "per subnet", "host" o "in trusted networks: `StrictHostKeyChecking=no`, `ForwardAgent=yes`
- Para obtener permiso sobre un archivo en forma octal **form**, el cual es útil para la configuración del sistema pero no disponible con `ls` y fácil de estropear, usando algo como

```
stat -c '%A %a %n' /etc/timezone
```

- Para selección interactiva de valores desde la salida de otro comando, usa https://github.com/mooz/percol`percol` o https://github.com/junegunn/fzf`fzf`.
- Para la interacción con archivos basados en la salida de otro comando (como `git`), usa `fpp` (https://github.com/facebook/PathPickerPathPicker).
- Para un servidor web sencillo para todos los archivos en el directorio actual (y subdirectorios), disponible para cualquiera en tu red, usa: `python -m SimpleHTTPServer 7777` (para el puerto 7777 y Python 2) y `python -m http.server 7777` (para 7777 y Python 3).
- Para ejecutar un comando con privilegios, usando `sudo` (para root) o `sudo -u` (para otro usuario). Usar `su` o `sudo bash` para realmente ejecutar un shell como este usuario. Usar `su -` para simular un login fresco como root u otro usuario.

0.1.4. Procesamiento de archivos y datos

- Para localizar un archivo por su nombre, en el directorio actual, `find . -iname *nombrearchivo*` (o similar). Para encontrar un archivo en cualquier lado, por su nombre, usar `locate algo` (Teniendo en mente `updatedb` encargado de indexar, quizas no se hayan indexado archivos creados recientemente).
- Por lo general buscar por la fuente o archivos de datos (más avanzado que `grep -r`), usar https://github.com/ggreer/the$_s ilver_s earcher$`ag`.$Paraconvertir HTML a texto$: `lynx -dump -stdin`
- Para Markdown, HTML, y todos los tipos de conversión de documentos, probar http://pandoc.org/`pandoc`.
- Si debes manipular XML, `xmlstarlet` es buena cosa aunque es viejo.
- Para JSON, usa `jq`.

- Para archivos Excel o CSV, https://github.com/onyxfish/csvkitcsvkit provee `in2csv`, `csvcut`, `csvjoin`, `csvgrep`, etc.

- Para Amazon S3, https://github.com/s3tools/s3cmd`s3cmd` es conveniente y https://github.com/bloomreach/s4cmd`s4cmd` es el mas rápido. Hecho por Amazon https://github.com/aws/aws-cli`aws` es esencial para otras tareas relacionadas al AWS.

- Conocer acerca de `sort` y `uniq`, incluyendo opciones de uniq `-u` y `-d` – ver unas lineas mas abajo.

- Conocer acerca de `cut`, `paste`, y `join`, sirve para manipular archivos de texto. Muchas personas usan `cut` pero se olvidan de `join`.

- Conocer acerca de `wc`, sirve para contar líneas (`-l`), caractéres (`-m`), palabras (`-w`) y bytes (`-c`).

- Conocer acerca de `tee`, sirve para copiar desde el stdin hacia un archivo y también hacia el stdout, así `ls -al | tee file.txt`.

- Conocer que locale afecta a muchas herramientas de línea de comandos de forma delicada, incluyendo el orden (compaginación) y rendimeinto. La mayoria de las instalaciones de Linux configuran `LANG` u otras variables de localización para la configuración local como US English. Pero estese alerta, el ordenamiento puede cambiar su localización. Y también las rutinas i18n pueden hacer que sort u otros comandos se ejecuten *muchas mas veces* y mas lentos. En algunas situaciones (tales como la realización de operaciones u operaciones singulares) puede, de forma segura ignorar, las rutinas lentas i18n por completo y utilizar el sort tradcional basado en byte, usando `export LC_ALL=C`.

- Conozca los esenciales `awk` y `sed` para mapeo de datos sencillo. Por ejemplo, sumar todos lo números en la tercera columna de un archivo de texto: `awk { x += $3 } END { print x }`. Esto es probablemente 3 veces más rápido y 3 veces más corto que su equivalente en Python.

- Para reemplazar todas las ocurrencias de una cadena en su lugar, en uno o más archivos:

```
perl -pi.bak -e 's/old-string/new-string/g' my-files-*.txt
```

- Para renombrar varios archivos a la vez de acuerdo a un patron, usar `rename`. Para renombramientos complejos, https://github.com/jlevy/reprenrepren debe ayudar.

```
# Recover backup files foo.bak -> foo:
rename 's/\.bak$//' *.bak
# Full rename of filenames, directories, and contents
foo -> bar:
repren --full --preserve-case --from foo --to bar .
```

- Usar `shuf` para mezclar o seleccionar líneas aleatorias desde un archivo.

- Conozca las opciones de `sort`. Para números, use `-n`, o `-h` para manipulación de números humanamente legibles (Ej. desde `du -h`). Conozca como funcionan principal de (`-t` y `-k`). En particular, este atento que lo necesitara para escribir`-k1,1` para ordenar por solo el primer campo; `-k1` significa ordenar de acuerdo a toda la línea. Orden estable (`sort -s`) puede ser útil. Por ejemplo, para organizar el primer por el campo 2, entonces secundariamente hacerlo por el campo 1, Puedes usar `sort -k1,1 | sort -s -k2,2`.
- Si necesitas escribir un tab literal en una línea de comandos en Bash (Ej. para el argumento -t de ordenar), presione **ctrl-v [Tab]** o escriba `$\t` (Este último es mejor porque puedes copiarlo/pegarlo).
- Las herramientas estandar para reparar el código fuente son `diff` y `patch`. Ver también `diffstat` para el resumen estadístico de un diff. Nota `diff -r` trabaja con directorios por completo. Usar`diff -r tree1 tree2 | diffstat` para obtener el resumen de cambios.
- Para archivos binarios, usar `hd` sobre sencillos "hex dumps" y `bvi` para edición de binario.
- También para archivos binarios, `strings` (además de `grep`, etc.) permite encontrar en el texto bits.
- Para diffs binarios (compresión delta), usar `xdelta3`.
- Para convertir la codificación del texto, probar `iconv`. O `uconv` para el uso avanzado; este soporta algunos elementos Unicode avanzados. Por ejemplo, este coloca en minúsculas y remueve todos los acentos (por expanción y colocandolos a ellos):

```
  uconv -f utf-8 -t utf-8 -x '::Any-Lower; ::Any-NFD;
[:Nonspacing Mark:] >; ::Any-NFC; ' < input.txt > output.txt
```

- Para dividir archivos en multiples partes, consultar `split` para dividir por tamaño y `csplit` para dividir por un patrón.
- Usar `zless`, `zmore`, `zcat`, y `zgrep` para operar sobre archivos comprimidos.

0.1.5. Depuración del sistema

- Para depuración web, `curl` y `curl -I` son útiles, o sus equivalentes `wget`, o el más moderno https://github.com/jakubroztocil/httpie`httpie`.
- Para el estado del disco/cpu/red, usar `iostat`, `netstat`, `top` (o mejor `htop`), y (especialmente) `dstat`. Estos comandos son buenos para obtener una idea rápida de lo que esta pasando en un sistema.
- Para una visión en mayor profundidad, se recomienda usar https://github.com/nicolargo/glances`glances`. Presenta varios niveles estadísticos del sistema en un solo terminal. Muy útil para una verificación rápida de varios subsistemas.
- Para conocer el estado de la memoria, ejecutar y entender la salida de `free` y `vmstat`. En particular, tener en cuenta que el valor "cached" es memoria mantenida por el kernel de Linux como un archivo de cache, por lo que efectivamente cuenta como valor "libre".

- El sistema de depuración de Java es harina de otro costal, pero un truco simple en Oracle y de otras maquinas virtuales de Java es ejecutar `kill -3 <pid>` y una traza completa de la pila (stack) y un resumen de la zona libre (heap) (incluyendo detalles de la colección de basura generacional, la que puede contener mucha información) serán descargados al stderr/logs.
- Usar `mtr` como un mejor seguidor de rutas (traceroute), para identificar problemas en la red.
- Para mirar porque el disco esta lleno, `ncdu` ahorra tiempo sobre comandos usuales como `du -sh *`.
- Para encontrar que socket o proceso esta utilizando el ancho de banda, prueba `iftop` o `nethogs`.
- La herramienta `ab` (viene con Apache) es útil para una verificación rapida y sucia del rendimiento del servidor web. Para pruebas de carga más complejas, prueba `siege`.
- Para depuración más seria de redes, `wireshark`, `tshark`, o `ngrep`.
- Investiga `strace` y `ltrace`. Estas son de utilidad si un programa esta fallando o colgándose, y no conoces por qué, o si quieres tener una idea general del rendimiento. Ten presente la opción de elaboración de perfiles (`-c`), y la habilidad de enganchar a un proceso en ejecución (`-p`).
- Investiga acerca de `ldd` para verificar librerías compartidas.
- Investiga cómo conectar a un proceso en ejecución con `gdb` y obtener su traza de pila.
- Usa `/proc`. Este es extremadamente útil a la hora de depurar problemas en vivo. Ejemplos: `/proc/cpuinfo`, `/proc/xxx/cwd`, `/proc/xxx/exe`, `/proc/xxx/fd/`, `/proc/xxx/smaps`.
- Cuando se depura porqué algo salio mal en el pasado, `sar` puede ser muy útil. Este muestra las estadísticas históricas de la CPU, memoria, red, etc.
- Para análisis mas profundos del sistema y rendimiento, ver `stap`

```
(\href{https://sourceware.org/systemtap/wiki}{SystemTap}),
\href{http://en.wikipedia.org/wiki/Perf_(Linux)perf}, y

\href{https://github.com/draios/sysdig}{\texttt{sysdig}}.
```

- Para confirmar en qué SO (Sistema Operativo) te encuentras utiliza `uname` o `uname -a` (información general Unix/kernel) o `lsb_release -a` (Linux distro info).
- Usar `dmesg` siempre que algo actúe de manera extraña (esto podría ser debido a problemas con el hardware o del driver).

0.1.6. Comandos concatenados

Algunos ejemplos de comandos concatenados (one-liners):

- Es remarcablemente útil en ocasiones en las que hay que realizar intersecciones, uniones, y diferenciaciones de archivos de texto via `sort`/`uniq`. Considere que `a` y `b` son archivos de texto que ya son únicos. Esto es rápido, y funciona con archivos de tamaño arbitrario, hasta varios gigabytes. (Sort no esta limitado por la memoria, aunque quizas necesite utilizar la opción `-T` si `/tmp` está en una pequeña partición

raíz.) Vea también la nota acerca `LC_ALL` y las opciones de `sort`, `-u` (dejadas de lado para clarificar mas abajo).

```
cat a b | sort | uniq > c #c la union de a y b
cat a b | sort | uniq -d > c #c es la intersección de a y b
cat a b b | sort | uniq -u >c #c es la diferencia entre a y b
```

- Usar `grep . *` para examinar visualmente todo el contenido de todos los archivos de un directorio, Ej. para directorios llenos de parámetros de configuración, como `/sys`, `/proc`, `/etc`.
- Sumar todos los números de la tercera columna de un archivo de texto (es probablemente 3 veces más rapido y con 3 veces menos código que el equivalente en Python):

```
awk '{ x += $3 } END { print x }' miarchivo
```

- Si quiere ver tamaños/fechas en un árbol de archivos, es como un `ls -l` recursivo pero es más fácil de leer que `ls -lR`:

```
find . -type f -ls
```

- Use `xargs` o `parallel` cuando pueda. Considere que puede controlar cuántos elementos son ejecutados por línea (`-L`) así como un paralelismo (`-P`). Si no está seguro de si lo hará correctamente, use primero xargs echo. También es práctico el `-I{}`. Ejemplos:

```
find . -name '*.py' | xargs grep some_function
cat hosts | xargs -I{} ssh root@{} hostname
```

- Digamos que tiene un archivo de texto, como un log de un servidor web, y un cierto valor empieza a aparecer en algunas líneas, como por ejemplo un parámetro `acct_id` que esta presente en el URL. Si quieres un recuento de cuantas peticiones hay por cada `acct_id`:

```
cat access.log | egrep -o 'acct_id=[0-9]+' |
cut -d= -f2 | sort | uniq -c | sort -rn
```

- Ejecta esta función para obtener un consejo aleatorio desde este documento (analiza el Markdown y extrae un elemento):

```
function taocl() {
  curl -s https://raw.githubusercontent.com/jlevy/
  the-art-of-command-line/master/README.md |
    pandoc -f markdown -t html |
    xmlstarlet fo --html --dropdtd |
    xmlstarlet sel -t -v "(html/body/ul/li[count(p)>0])
    [$RANDOM
    mod last()+1]" |
    xmlstarlet unesc | fmt -80
}
```

0.1.7. Oscuro pero útil

- `expr`: ejecuta operaciones aritméticas o booleanas o evalúa expresiones regulares.
- `m4`: macro procesador sencillo.
- `yes`: imprime un string sin fin.
- `cal`: bonito calendario.
- `env`: ejecuta un comando (útil en scripts).
- `printenv`: imprime las variables del entorno (útil en depuración y scripts).
- `look`: busca palabras en inglés (o líneas en un archivo) comenzando con una cadena.
- `cut`, `paste` y `join`: manipulación de datos.
- `fmt`: formato de texto de párrafo.
- `pr`: formato de texto en páginas/columnas.
- `fold`: envolturas de líneas de texto.
- `column`: formato de texto en columnas o tablas.
- `expand` y `unexpand`: convertidor entre tabs y espacios.
- `nl`: agrega números de línea.
- `seq`: imprime números.
- `bc`: calculadora.
- `factor`: factorización de números enteros.
- `gpg`: cifrado y firmas digitales.
- `toe`: tabla de información de términos.
- `nc`: depuración de red y transferencia de datos.
- `socat`: socket relay y redireccionador de puerto tcp (similar a `netcat`).
- `slurm`: visualización del tráfico de red.
- `dd`: moviliza datos entre archivos y dispositivos.
- `file`: identifica el tipo de archivo.
- `tree`: muestra directorios y subdirectorios como un árbol anidado; parecido a `ls` pero recursivo.
- `stat`: información del archivo.
- `tac`: imprime archivos de forma inversa.

- `shuf`: selección de líneas de un archivo de forma aleatorea.
- `comm`: compara archivos ordenados línea por línea.
- `pv`: monitor del progreso de datos, a través, de una tubería.
- `hd` y `bvi`: descarga o edita archivos binarios.
- `strings`: extrae textos de archivos binarios.
- `tr`: traducción y manipulación de caracteres.
- `iconv` o `uconv`: conversión de codificaciones de texto.
- `split` y `csplit`: división de archivos.
- `sponge`: lee todas las entradas antes de escribirlo, útil para lectura y luego la escritura hacia el mismo archivo, Ej., `grep -v something some-file | sponge some-file`
- `units`: unidades de conversión y calculo; convierte furlongs(estadios) por fortnight(quincenas) para twips por blink (ver también `/usr/share/units/definitions.units`)
- `7z`: compresión de archivos de alto nivel.
- `ldd`: información de libreria dinámica.
- `nm`: archivo de objeto de símbolos.
- `ab`: testeo (benchmarking) de servdores web.
- `strace`: depuración de llamadas del sistema.
- `mtr`: mejor traceroute para la deputación de la red.
- `cssh`: shell concurrente visual.
- `rsync`: sincronización de archivos y carpetas sobre SSH.
- `wireshark` y `tshark`: captura de paquetes y depuración de la red.
- `ngrep`: grep para la capa de la red.
- `host` y `dig`: consulta DNS.
- `lsof`: descriptor de archivo de procesos e información de socket.
- `dstat`: sistema de estadísticas útil.
- https://github.com/nicolargo/glances`glances`:vistazo de multi-subsistemas, de alto nivel.
- `iostat`: estadísticas del CPU y uso del disco.
- `htop`: version mejorada de top.

- `last`: historial de login.
- `w`: ¿quién esta authenticado?.
- `id`: información de identidad de usuario/grupo.
- `sar`: sistema de estadísticas histórico.
- `iftop` o `nethogs`: utilización de la red por un socket o process.
- `ss`: estadistícas de socket.
- `dmesg`: arranque y sistema de mensajes de error.
- `hdparm`: manipulación/rendimiento de discos SATA/ATA.
- `lsb_release`: información de la distribución de Linux.
- `lsblk`: Lista de bloques de dispositivos: un árbol de vista de sus discos y particiones de disco.
- `lshw`, `lscpu`, `lspci`, `lsusb`, `dmidecode`: información de hardware, incluyendo CPU, BIOS, RAID, grafícos, dispositivos, etc.
- `fortune`, `ddate`, y `sl`: um, bien, este depende si tiene la consideración de locomotoras de vapor y citas Zippy "práctico".

0.1.8. Solo MacOS

Estos son los elementos relevantes *solo* en MacOS.

- La gestión de paquetes con `brew` (Homebrew) y/o con `port` (MacPorts). Estos pueden ser usados para instalar en MacOS muchos de los comandos anteriores.
- Copiar la salida de cualquier comando para una aplicación desktop con `pbcopy` y pegar desde una entrada con `pbpaste`.
- Para abrir un archivo con una application desktop, usar `open` o `open -a /Applications/Whatever.app`.
- Spotlight: Busca archivos con `mdfind` y listar metadata (como la información EXIF de las fotos) con `mdls`.
- Tenga en cuenta que MacOS esta basado en UNIX BSD, y cualquier comando (por ejemplo `ps`, `ls`, `tail`, `awk`, `sed`) tiene variaciones sutiles con respecto a las de Linux, que esta muy influenciado por el estilo System V de Unix y las herramientas GNU. A menudo se puede ver la diferencia en las páginas man al incluir el encabezado "BSD General Commands Manual." En algunos casos también se puede instalar versiones de GNU (tales como `gawk` y `gsed` para awk y sed de GNU). Si escribe scripts multiplataforma en Bash, evita tales comandos (por ejemplo, considerando Python o `perl`) o probarlos cuidadosamente.

0.1.9. Más recursos

- awesome-shell[https://github.com/alebcay/awesome-shell] : Una lista completa de herramientas de la consola y recursos
- Strict mode [http://redsymbol.net/articles/unofficial-bash-strict-mode/] para escribir mejores programas

0.1.10. Advertencia

Con excepción de tareas muy pequeñas, el código se escribe para que otros puedan leerlo. Un poder conlleva una responsabilidad. El hecho de que puedas hacer algo en Bash, no implica necesariamente que debas hacerlo! ;)

Más sobre este tema: http://www.quora.com/What-are-some-lesser-known-but-useful-Unix-commandsoriginally http://www.quora.com/What-are-the-most-useful-Swiss-army-knife-one-liners-on-Unixappeared en http://www.quora.com/What-are-some-time-saving-tips-that-every-Linux-user-should-knowQuora, pero dado el interés mostrado ahí, parece que no es del todo malo usar Github, donde personas con mayor talento pueden facílmente sugerir mejoras. Si ve un error o algo que pueda mejorarse, por favor cree un aviso o PR! (Por supuesto revise la sección meta de PRs/avisos primero.)

0.1.11. Licencia