Javier Gonzalez

[Nombre de la compañía]

[Fecha]

An holistic proposal for managing the command line



disclaimer

Given that my mother tongue is Spanish (one of the most used in the world) and for accuracy this document and all related ones are written in Spanish.

However, if you are interested, do not hesitate to mail me, I will be happy to translate it into English for you.

abstract

En este documento se plantea una nueva forma de indicar los argumentos aceptados por un programa a través de la línea de comandos de una manera homogénea y común a cualquier Sistema Operativo.

También se propone la creación y uso de una librería estándar de manejo de la línea de comandos homogénea para cada lenguaje de programación

1 Introducción 5

2 Taxonomía de la línea de comandos 7

2.1 Clasificación 7

2.2 Tipología 8

3 Situación actual 11

4 Propuesta 13

4.1 Librería 13

# Introducción

El problema se presenta cada vez que es necesario desarrollar un nuevo programa

un nuevo desarrollo; que Cualquier programa susceptible de ser ejecutado a través de una linea de comandos necesita una serie de parámetros de entrada para su ejecución, de lo contario siempre generaría el mismo resultado.

Ahora bien, dependiendo del sistema operativo, la forma habitual de indicar estos parámetros varía; es mas, cada programador podría definir un nuevo sistema de entrada de parámetros. Esto obliga, o al menos debería, a que un programa multiplataforma debe adecuarse a cada sistema operativo y controlar programáticamente el estilo de cada uno de ellos.

Veamos algunos ejemplos:

\*\*Sistemas \*nix\*\*

Tipicamente estos sistemas utilizan uno o dos guiones: \*\*'-'\*\* o \*\*'--'\*\*; para indicar un parámetro, donde:

- La primera forma se aplica a parámetros definidos por un solo carácter y permite además concatenarlos

- La segunda forma se entiende como parámetro extendido definido por una palabra

Ejemplos:

`command -h`

`command --help`

`command -af`

`command -a -f`

`command -o output\_file`

\*\*Sistemas Windows\*\*

En estos sistemas, la linea de comandos \*\*no es sensible a mayúsculas/minúsculas\*\* y se utiliza el caracter \_\_slash\_\_ '\*\*/\*\*'. Y también añade diferentes formas de indicarlos.

Estos son algunos ejemplos:

`command /h`

`command /-Y`

`command /D:value`

`command /o output\_file`

Por otro lado, con la inclusión de `PowerShell` se han añadido también el uso de guiones y algunas otras variantes en función de cada comando; algunos especificados en modo \_\_camel case\_\_ aunque como se ha indicado, en este sistema no se distinguen mayúsculas de minúsculas.

\*\*zOS y otros\*\*

Otros sistemas operativos soportan otros métodos. Por ejemplo zOS usa /\_characters\_, donde \_characters\_ es el mínimo numero de caracteres necesario para identificar un comando.

Por ejemplo `/pre pattern` y `/preffix pattern` son equivalentes en el sentido de que no existe mas que un comando que empiece por la secuencia `pre`

## Conclusiones e inefciencias

Sea el sistema operativo que sea, el desarrollador debe \*\*siempre\*\* programar la gestión de estos parámetros para cada uno de los programas que desarrolle (soy consciente de que existen algunas utilidades que facilitan un poco la tarea, pero suelen ser incompletas) a pesar de ser una de las tareas comunes de cualquier programa, lo cual conlleva un esfuerzo de trabajo innecesario y repetitivo.

Con estos sistemas se plantean los siguientes problemas:

Dado un parametro P o Parameter:

- ¿Como podemos saber si es un flag o una opcion?

- Si es un flag, ¿Activa o desactiva esa funcionalidad? (Vease el caso -verbose y -silent)

- ¿Por que tengo que elegir como usuario, y controlar como desarrollador, las dos opciones -P y --Parameter?

- En el caso de que no exista una version corta del parámetro, ¿Por que necesito escribir Parameter si con la primera letra ya seria suficiente?

- Cuanto mas caracteres tenga que escribir mas posible es que introduzca un error de tipografia

# Taxonomía de la línea de comandos

Por definición, un programa acepta un conjunto de parámetros y argumentos para realizar el proceso para el que fue diseñado (por ahora utilizaremos la palabra parámetro de manera genérica y más adelante matizaremos este concepto), por ejemplo, MS Word acepta, en su versión 365, 18 parámetros and switches además de un conjunto indefinido de “variables personalizadas”

El primer hecho a tener en cuenta es que **todos los parámetros que acepta el programa se conocen en tiempo de diseño**; por lo tanto, está controlados y codificados en el propio código, aceptando de manera implícita o explícita un valor por defecto para los casos en los que no se indique otro valor de manera explícita.

El segundo hecho a tener en cuenta es que **la línea de comandos se entrega al programa como una secuencia de caracteres**; dicho de otro modo, el “lanzador” no realiza ningún proceso sobre ella, dejando la labor de verificar la corrección lógica y funcional de los parámetros al propio programa.

Por último, y obviando los programas con interfaz gráfico, las únicas maneras de modificar esos valores por defecto para cada ejecución especifican son:

1. A través de un fichero de configuración

2. A través de variables de entorno

3. En la propia línea de comandos

En cualquiera de los tres casos, el programa es responsable de:

1. Implementar la funcionalidad correspondiente: fichero, entorno, línea de comando

2. Realizar la validación lógica; es decir, que parámetros son válidos y cuales no

3. Realizar la validación funcional; es decir, verificar que el dato pasado es correcto desde el punto de vista de tipo de dato y valor del mismo.

## Clasificación

Desde el punto de vista de los parámetros se pueden clasificar en cuatro tipos XXXXX:

* Switches
* Opciones
* Argumentos
* Definiciones

Switches

Parámetros binarios; es decir, no necesitan información adicional, el propio parámetro define si activa o inactiva alguna funcionalidad dentro del programa

Un ejemplo típico sería, en entornos \*nix, el switch -v o –verbose que normalmente se asocia con la generación de mas o menos información acerca del progreso del programa.

Opciones

Son parámetros que necesitan información adicional para quedar definidos, XXX

Argumentos

Los objetos que deben ser procesados por el programa.

Definiciones

A falta de una definición mejor, damos en llamar definiciones a aquellos parámetros que pueden ser aceptados por el programa para su posterior uso en tiempo de ejecución pero que no están codificados en el propio programa.

XXXX

Ejemplo:

Supongamos un programa `sample` cuya funcionalidad es: Lee un fichero y lo escribe en otro sitio cambiando ciertas palabras por otras en función de lso datos indicados por el usuario. Su ayuda podría mostrar algo similar a lo siguiente:

```

sample [-h] [-v] [-i input\_dir] [-o output\_dir] [[-d palabra=valor] ...] [...]

-h: Esta ayuda

-v: Mostrar información de progreso

-i input\_dir: Directorio de entrada

-o output\_dir: Directorio de salida. Si se omite se usara la salida estandar

-d palabra=valor: conjunto de palabras a cambiar en el fichero

...: Archivos a procesar. Si se omite se utilizará la entrada estandar

```

En este programa:

- `-h`, `-v` son \*\*Flags\*\*

- `-i`,`-o` son \*\*Opciones\*\*

- `-d` son \*\*Definiciones\*\*

- `...` son los \*\*Argumentos\*\* a procesar

## Tipología

Independientemente del modo de especificar los parámetros, como ya se ha indicado, estos son recibidos por el programa como cadenas de caracteres, pero en función del tipo de opción su valor asociado debe ser convertible a un tipo de dato concreto y/o cumplir con alguna posible serie de restricciones: un rango de valores, un conjunto de valores, etc.

De acuerdo con esto, cualquier parámetro pertenecerá a una de los siguientes grupos:

|  |  |
| --- | --- |
| Grupo | Descripcion |
| Caracteres | Un simple carácter o una cadena de caracteres |
| Números | Un valor numérico, posiblemente con ciertas limitaciones respecto a su valor (positivo, entero, dentro de un rango, etc.) |
| Fechas y horas | En cualquiera de sus formatos: fecha, fecha hora, timestamp |
| Almacenamiento | Directorios, ficheros, red, etc. |
| Lógico | Una secuencia que pueda interpretarse como un valor lógico |
| Configuración | Como caso especial de almacenamiento, un archivo que contiene una secuencia de parámetros |

|  |  |  |
| --- | --- | --- |
| Grupo | Tipo | Descripcion |
| Cadenas | |  |
|  | CHAR | Un carácter |
|  | STRING | Una cadena |
| Numeros | |  |
|  | BYTE | Un número en los rangos [0,255] o [-127,127] |
|  | SHORT | Un número en los rangos [0,65535] o [-32768, 32767] |
|  | NATURAL | Un número positivo hasta 4294967295 |
|  | INTEGER | Un número entero dependiendo de la arquitectura |
|  | LONG | Un número entero dependiendo de la arquitectura |
|  | DECIMAL | Un número con decimales |
| Fechas y horas | |  |
|  | DATE | Una fecha de acuerdo con el formato del país |
|  | TIME | Una hora en formato hh:mm:ss |
|  | DATETIME | Fecha y hora en formato “YYYY/MM/DD hh:mm:ss” |
|  | TMS | Timestamp |
| Almacenamiento | |  |
|  | DIR | Una ruta que pueda intepretarse como un directorio |
|  | DIR\_EXISTS | Un ruta a un directorio existente |
|  | FILE | Una ruta que pueda intepretarse como un fichero |
|  | FILE\_EXISTS | Una ruta a un fichero existente |
| Lógico | |  |
|  | BOOL | Una secuencia que pueda interpretarse como un valor lógico:  TRUE / FALSE  ON / OFF  YES / NO  -1 / 0 / 1 |
| Configuración | |  |
|  | CONFIG | Un fichero de configuración |
|  | CONFIG\_RELAXED | Un fichero de configuración |

Es importante señalar que, esta tipologia esta codificada expresamente en todos y cada uno de los programas

## Definiciones

Introduzcamos algunas definiciones:

\*\*Parametro\*\* o \*\*argumento\*\* son sinónimos y es cualquier elemento que aparece después del comando en si mismo

Ahora bien, estos parámetros pueden ser de tres tipos:

1. Datos de entrada

2. Opciones

3. Flags

Y cada programa puede aceptar un determinado número de opciones, incluida ninguna, y de flags, incluidos ninguno, pero en el caso de que exista al menos uno, este, por definición debe asumir un valor por defecto cuando no es especificado en la linea de comandos.

### Datos de entrada

Es cualquier elemento o secuencia de ellos que debe ser procesado por el programa.

Por ejemplo: `cat fichero.txt` mostrara por pantalla el contenido del fichero fichero.txt

### Opciones

Es cualquier elemento o secuencia de ellos que modifica o afecta al comportamiento del programa aportándole algun tipo de información

Por ejemplo: `grep -e \_pattern\_` o `grep --regexp \_pattern\_` indica a grep que patrón usar

### Flags

Es cualquier elemento que activa o desactiva alguna característica o funcionalidad del programa.

El caso mas típico seria el flag: `-h` o `--help` que indica al programa que muestre su ayuda

Otros ejemplos podrían ser:

`command -v` o `command --verbose` que suele usarse para generar información de progreso

`command -s` o `command --silent` que suele usarse para evitar que se genere esa información

Notese que estos flags son complementarios; es decir, afectan a las misma funcionalidad.

# Situación actual

Ahora bien, dependiendo del sistema operativo, la forma habitual de indicar estos parámetros varía; es mas, cada programador podría definir un nuevo sistema de entrada de parámetros. Esto obliga, o al menos debería, a que un programa multiplataforma debe adecuarse a cada sistema operativo y controlar programáticamente el estilo de cada uno de ellos.

Veamos algunos ejemplos:

\*\*Sistemas \*nix\*\*

Tipicamente estos sistemas utilizan uno o dos guiones: \*\*'-'\*\* o \*\*'--'\*\*; para indicar un parámetro, donde:

- La primera forma se aplica a parámetros definidos por un solo carácter y permite además concatenarlos

- La segunda forma se entiende como parámetro extendido definido por una palabra

Ejemplos:

`command -h`

`command --help`

`command -af`

`command -a -f`

`command -o output\_file`

\*\*Sistemas Windows\*\*

En estos sistemas, la linea de comandos \*\*no es sensible a mayúsculas/minúsculas\*\* y se utiliza el caracter \_\_slash\_\_ '\*\*/\*\*'. Y también añade diferentes formas de indicarlos.

Estos son algunos ejemplos:

`command /h`

`command /-Y`

`command /D:value`

`command /o output\_file`

Por otro lado, con la inclusión de `PowerShell` se han añadido también el uso de guiones y algunas otras variantes en función de cada comando; algunos especificados en modo \_\_camel case\_\_ aunque como se ha indicado, en este sistema no se distinguen mayúsculas de minúsculas.

\*\*zOS y otros\*\*

Otros sistemas operativos soportan otros métodos. Por ejemplo zOS usa /\_characters\_, donde \_characters\_ es el mínimo numero de caracteres necesario para identificar un comando.

Por ejemplo `/pre pattern` y `/preffix pattern` son equivalentes en el sentido de que no existe mas que un comando que empiece por la secuencia `pre`

## Alternativas

Python optparse

R optparse

Unix getopt

## Conclusiones e ineficiencias

Sea el sistema operativo que sea, el desarrollador debe \*\*siempre\*\* programar la gestión de estos parámetros para cada uno de los programas que desarrolle (soy consciente de que existen algunas utilidades que facilitan un poco la tarea, pero suelen ser incompletas) a pesar de ser una de las tareas comunes de cualquier programa, lo cual conlleva un esfuerzo de trabajo innecesario y repetitivo.

Con estos sistemas se plantean los siguientes problemas:

Dado un parametro P o Parameter:

- ¿Como podemos saber si es un flag o una opcion?

- Si es un flag, ¿Activa o desactiva esa funcionalidad? (Vease el caso -verbose y -silent)

- ¿Por que tengo que elegir como usuario, y controlar como desarrollador, las dos opciones -P y --Parameter?

- En el caso de que no exista una version corta del parámetro, ¿Por que necesito escribir Parameter si con la primera letra ya seria suficiente?

- Cuanto mas caracteres tenga que escribir mas posible es que introduzca un error de tipografia

# Propuesta

El siguiente esquema propone resolver estas "deficiencias" y clarificar de cara al usuario los significados de cada parámetro:

1. Un flag se prefija siempre con los signos + o - y su significado es \*\*siempre\*\* el mismo: + activa la funcionalidad y - la desactiva.

2. Dado que los flags no necesitan información asociada se pueden concatenar en una secuencia de caracteres con la unica condicion de que ninguno de ellos empiece por el mismo caracter

3. Una opcion se prefija \*\*siempre\*\* con la barra / y su valor asociado se establece en la siguiente palabra

4. Cualquier parámetro queda definido en el momento en el que no existe duda acerca de la intención del usuario; es decir, no es necesario introducir el nombre completo del parámetro

Por ultimo, y como consejo, el manual de usuario o la ayuda en linea, deberia indicar los valores por defecto de todos los flags y opciones soportadas por el programa

## Librería

De acuerdo con lo expuesto, ya estamos en condiciones de definir los requerimientos para una librería de gestión de la línea de comandos independiente del lenguaje:

El analizador debe preservar las instrucciones del usuario(2), funcionar de manera segura en un entorno multihilo(1) y garantizar que los datos introducidos son válidos lógica y funcionalmente(3)

## Entorno multihilo: Threadsafe

Este requerimiento implica que

## Preservar instrucciones del usuario

Este requerimiento implica que

## Validez lógica y funcional

1. Preservar las instrucciones del usuario implica que: