ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

УДК 004.04

СОГЛАШЕНИЯ «POSIX» ПО ОБРАБОТКЕ ОПЦИЙ И АРГУМЕНТОВ КОМАНДНОЙ СТРОКИ

О.В. СУХОРУКОВ
(Представлено: канд. физ.-мат. наук, доц. О.В. ГОЛУБЕВА)

Рассматривается стандарт «POSIX» касательно соглашений по обработке аргументов командной строки, которые можно рассматривать как совокупность договоренностей между разработчиком операционной системы и прикладным программистом. Представлены основные определения соглашений, приводятся примеры использования, комментарии автора.

Работа с командной строкой – это тот базис, который предоставляет гибкость и мощь UNIX/Linux систем, а соблюдение соглашений «POSIX» (Portable Operating System Interface) касательно опций и аргументов командной строки облегчает и стандартизирует процесс взаимодействия с командной оболочкой – поведение незнакомой программы на конкретную команду становится ожидаемым.

Компьютерная программа – это последовательность инструкций, написанных на одном или более языке программирования и предназначенных для исполнения вычислительной машиной [1]. Программа может получать данные извне. На начальном этапе исполнения основным способом получения данных может служить обработка аргументов командной строки.

В программах, созданных на линейке языков «C» таким способом, является чтение аргументов функции «main()» [2, с. 58], которая служит в качестве начальной точки исполнения программы [3]. В этом случае данные называют «аргументами программы», которые не стоит отождествлять с «аргументами командной строки». Это разные понятия. Дело в том, что программа получает свои аргументы от родительского процесса, в качестве которого не всегда выступает командная оболочка. Таким процессом, к примеру, может оказаться браузер, в котором в настоящее время никакой «командной строки» нет [2, с. 58].

В момент запуска программы функции «main()» передаются два аргумента. В первом, обычно называемом «argc» (сокращение от *argument count*), стоит количество аргументов, заданных в командной строке. Второй параметр, называемый «argv» (от *argument vector*), является указателем на массив символьных строк, содержащих сами аргументы. Для работы с этими строками обычно используются указатели нескольких уровней. По общепринятому соглашению «argv[0]» – это имя вызываемой программы, так что значение «argc» никогда не бывает меньше 1 [4, с. 150].

Аргументом командной строки может служить любой набор символов. Между собой аргументы обычно разделяются пробелами либо символами табуляции, за исключением случаев, когда разделитель экранирован кавычками [3]. Несмотря на то, что организация и дальнейшее использование аргументов строго не регламентируются и могут быть ограничены лишь фантазией программиста, в «UNIX/Linux» среде существует общепринятая договоренность с набором рекомендаций по их структуре и применению.

Такой набор рекомендаций описывается в так называемом стандарте «POSIX» (*Portable Operating System Interface*), который можно рассматривать как совокупность договоренностей между разработчиком операционной системы и прикладным программистом. «Договоренность» означает, прежде всего, одинаковость интерпретации (семантики) слов и выражений [5]. Помимо обработки командной строки стандарт «POSIX» описывает множество базовых, системных сервисов, необходимых для функционирования прикладных программ, доступ к которым предоставляется посредством интерфейса, специфицированного для языка «C», командного языка и общеупотребительных служебных программ [6; 7].

«POSIX» не ограничен рамками «UNIX/Linux» среды. Существуют операционные системы (ОС) «независимого происхождения» (например, системы реального времени), предоставляющие необходимые сервисы и тем самым поддерживающие выполнение «POSIX-совместимых» приложений. Можно утверждать, что следование стандарту «POSIX» облегчает перенос приложений практически на любую операционную платформу. При этом дополнительные усилия по повышению мобильности, прилагаемые на этапе разработки, безусловно, окупаются [6].

В первую очередь, стандарт «POSIX» разделяет аргументы командной строки на «опции» и «операнды» (независимые аргументы).

«Опции» являются специальными управляющими аргументами командной строки, которые призваны изменять поведение программы или предоставлять программе дополнительную информацию. По старому соглашению, опции начинаются с дефиса (знак минус) и состоят из единственной буквы [8, с. 38]. Это

так называемые короткие опции. В новом соглашении также существует поддержка длинных опций, начинающихся с двух, подряд стоящих дефисов (знаков минусов). Длинная опция может состоять из любого количества символов, исключающих символ разделитель [7].

«Операндом» в этом случае считаются все остальные аргументы командной строки, не являющиеся «опциями». Имя вызываемой программы «argv[0]» функции «main()» также является «операндом».

Основные перефразированные соглашения стандарта «POSIX» по опциям и аргументам (операндам) [7] представлены в таблицах 1 и 2.

Таблица 1 – соглашения «POSIX» по коротким опциям и аргументам

№	POSIX	Пример	Комментарий
1	Имя опции должно быть простым буквенно-цифровым символом. Опции с множеством цифр не допускаются	-a -b -c -1 -2 -3	Опция «-W» зарезервирована для производителей, реализующих утилиты «POSIX»
2	Все опции должны начинаться с символа «-»		Символ «-» отличает опцию от аргумента. (Последовательность из двух символов «--» отличает длинную опцию от короткой)
3	Для опций, не требующих аргументов (операндов), должно быть возможно объединение нескольких опций после единственного символа «-»	foo -a -b -c foo -abc	Оба примера должны интерпретироваться одинаково
4	Если опция требует аргумент, он должен быть отделен от опции пробелом	fgrep -f patfile	На практике это соглашение применяется редко – опция и ее аргумент (операнд) обычно могут находиться в одной строке: «fgrep -fpatfile» (такое поведение реализуется функциями «getopt()» и «getopt_long()»)
5	Аргументы опций не должны быть необязательными	fgrep -f <patfile>	Это означает, что если в документации программы указано, что опции требуется аргумент, этот аргумент должен присутствовать всегда, иначе программа потерпит неудачу. Однако, как и с предыдущим пунктом, это одно из тех соглашений, которое не всегда поддерживается. К примеру, «GNU getopt()» предусматривает необязательные аргументы опций, поскольку считает, что они могут быть полезны
6	Если опция принимает аргумент, который может иметь несколько значений, программа должна получать этот аргумент в виде одной строки со значениями, разделенными запятыми или иным установленным разработчиком разделителем	prog -u pasha,masha,dasha prog -u «pasha masha dasha»	В первом примере разделение осуществляется запятыми, во втором – пробелами, аргументы опции в этом случае экранируются кавычками.
7	Опции должны находиться в командной строке первыми, перед аргументами (операндами)	prog -abc arg1 arg2	Версии «getopt() Unix» придерживаются этого соглашения, версии «GNU getopt()» по умолчанию этого не делают, однако это можно настроить
8	Порядок, в котором приведены опции, не должен играть роли. Однако, для взаимно исключающих опций, когда одна опция перекрывает установки другой, последняя считается главной	prog -abc prog -bca prog -cab	Все примеры должны интерпретироваться одинаково
9	Порядок, в котором приведены аргументы, имеет для программы значение	prog arg1 arg2 arg3	Каждая программа должна четко документировать последовательность расположения используемых аргументов
10	Программы, читающие или записывающие именованные файлы, должны трактовать единственный аргумент «-» как означающий стандартный ввод или стандартный вывод, в зависимости от того, что подходит программе		Если операнд задает читаемый или записываемый файл, то знак минус на его месте используется только для обозначения стандартного ввода (или стандартного вывода, если из контекста ясно, что специфицируется выходной файл)

Следует отметить, что даже многие стандартные программы не следуют всем указанным соглашениям. Главной причиной является историческая совместимость, многие такие программы предшествовали систематизации этих соглашений [8, с. 40].

«Длинные опции» – это естественное развитие стандарта «POSIX» в ответ на тенденции развития информационных технологий. Компьютеры стали быстрее, память увеличилась в геометрической прогрессии, возможности программ стали шире, а алгоритмы сложнее [9]. Отпала необходимость в экономии места, появились технологии интеллектуального дополнения ввода, при которых нивелировалась разница в скорости ввода между короткими опциями и их длинными соотношениями. Длинные опции легче запомнить, и они предоставляют возможность стандартизировать их применение среди всех утилит GNU. Хорошим примером будет являться опция «--help», которая обычно не изменяет своего значения среди утилит GNU, чего нельзя сказать по поводу ее сокращенной версии «-h».

Поскольку длинные опции начинаются с «--», совместное применение их с короткими опциями не конфликтует с предшествующими соглашениями «POSIX» [7]. Длинные опции GNU имеют свои собственные соглашения, реализованные в функции «getopt_long()». Основные перефразированные соглашения стандарта «POSIX» по длинным опциям представлены в таблице 2.

Таблица 2 – соглашения «POSIX» по длинным опциям

№	POSIX	Пример	Комментарий
1	Каждая коротка опция (один символ) должна иметь также свой вариант в виде длинной опции	-h --help	Оба примера должны интерпретироваться одинаково
2	Дополнительные специфические для GNU опции не нуждаются в соответствующей короткой опции	--trulala	В документации к стандарту «POSIX» в разделе 4.9 прилагается таблица стандартизированных длинных опций по утилитам GNU [7]. Опция, не входящая в эту таблицу, считается специфической для GNU (см. пример) и не нуждается в соответствующей короткой опции, однако тем же стандартом «POSIX» рекомендуется ее создать
3	Длинную опцию можно сократить до кратчайшей строки, которая остается уникальной	--verbose (--verbo) --verbatim (--verba)	В примере приложение регламентирует две опции «--verbose» и «--verbatim». Стандартом «POSIX» допускается при использовании опций сокращать их имена до минимального значения, про которых программа сможет их идентифицировать. Минимальные допустимые значения в этом случае будут «--verbo» и «--verba» соответственно
4	Аргументы опций могут быть необязательными		Для таких опций считается, что аргумент присутствует, если он находится в одной строке с опцией. Однако в существующей реализации «getopt_ long()» это работает лишь для коротких опций. Например, если «-x» такая опция и дана строка «foo -xYANKEES -y», аргументом «-x» является «YANKEES». То для «foo -x -y», у опции «-x» нет аргументов [8, с. 40]
5	Программы могут разрешить длинным опциям начинаться с одного символа «-»		Такое поведение типично для многих программ X Window [8, с. 40]. По мнению автора, такое допущение в соглашении идет в разрез с соглашениями «POSIX» по коротким опциям и является грубейшей ошибкой
6	Специальный элемент командной строки «--», который не является ни опцией, ни операндом, обозначает конец опций. Все последующие слова трактуются как операнды, даже если они начинаются со знака минус	prog -- -a -b --opt	В текущем примере «-a», «-b», «--opt» интерпретируются не как опции, а не как аргументы

Проанализировав соглашения в таблицах 1, 2, можно подытожить сказанное выше следующим выводом: опции согласно стандарту «POSIX» – это унифицированный интерфейс для передачи данных в программу через аргументы [2, с. 59].

Механизм опций стандарта «POSIX» позволяет разделить все аргументы программы на следующие категории:

1 Опции – различают:

- короткие (односимвольные);
- длинные (многосимвольные).

2 Зависимые аргументы (аргументы опций) – это аргументы, наличие которых у опции предполагается. Например, имя файла, которое указывается вслед за опцией «-o» компилятора «gcc», является зависимым аргументом. Опции, не требующие наличия зависимых аргументов, называют флагами. Как правило, одна опция может принимать только один такой аргумент.

3 Свободные аргументы – это аргументы, которые не связаны напрямую с опциями. Например, имя каталога, передаваемое программе «rmdir», является свободным аргументом.

Процедура обработки командной строки считается тривиальной задачей. Массив «argv» содержит аргументы командной строки, «argc» указывает, сколько их было передано, а стало быть для работы с командной строкой программам вполне достаточно параметров функции «main()». Но только до тех пор, пока программист не решит следовать соглашениям «POSIX».

Анализ таблиц 1, 2 раскрыл огромное количество нюансов, которое необходимо реализовать, это и объединение опций, и разделение опций на длинные и короткие, контроль обязательных и необязательных аргументов и т. п.

Еще примерно в 1980-х годах группа поддержки «UNIX» для «System III» в «AT&T» заметила, что каждая программа «UNIX» использовала для разбора аргументов свои собственные методики. Чтобы облегчить работу программистов в упрощении написания кода, придерживающегося стандартных соглашений, была разработана функция «getopt()». Функция GNU «getopt_long()» предоставляет совместимую с «getopt()» версию, а также упрощает разбор длинных опций в описанной ранее форме [8, с. 43]. Однако эти функции до сих пор являются актуальными и используются практически без изменений. Первоначально функции были разработаны для «C», для использования в других языках, таких как «java», «Perl», «Python», «PHP» и др., существуют соответствующие оболочки, так называемые фреймворки над функциями, позволяющие задействовать их в своем коде [10].

Подавляющее большинство ПО в UNIX/Linux среде используют для своей работы аргументы командной строки. Даже программы с графическим интерфейсом, которым, казалось бы, и вовсе не нужна консоль, интерпретируют десятки входных параметров. Это может быть задание начальных настроек, включение отладочного режима или просто вывод версии программы без ее запуска [10]. И это не только дань традиции, такой подход позволяет обмениваться выходными данными [11], использовать функционал других программ, работать с программой через «голую» консоль, не имея графического окружения, объединять, дополнять, и даже изменять функционал программ, не изменяя их кода [12]. То есть работа с командной строкой это и есть тот базис, который предоставляет гибкость и мощь UNIX/Linux систем, а соблюдение соглашений «POSIX» касательно опций и аргументов командной строки, облегчает и стандартизирует процесс взаимодействия с командной оболочкой – поведение незнакомой программы на конкретную команду становится ожидаемым.

ЛИТЕРАТУРА

1. Интернет, компьютеры, софт и прочий Hi-Tech [Электронный ресурс] // Что такое компьютерная программа: сайт. – Электрон. текстов. дан. – [Б. м.], 2010. – Режим доступа: http://xbb.uz/soft/Chto_takoe_kompjuternaja_ programma. – Загл. с экрана.
2. Иванов, Н.Н. Программирование в Linux [Текст] : самоучитель / Н.Н. Иванов. – 2-е изд., перераб. и доп. – СПб. : БХВ-Петербург, 2012. – 400 с.
3. MSDN [Электронный ресурс] // Функция main и выполнение программ: сайт. – Электрон. текстов. дан. – [Б. м.], 2016. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/3ze4ytsc.aspx. – Загл. с экрана.
4. Керниган, Б. Язык программирования Си [Текст] : учеб. метод. пособие / Б. Керниган, Д. Ритчи; пер. с англ. – 3-е изд., испр. – СПб. : Невский Диалект, 2001. – 352 с.
5. Романюк, С. Сюрпризы POSIX : сайт / С. Романюк // Журнал. Открытые Системы [Электронный ресурс]. – Электрон. текстов. дан. – [Б. м.], 1999. – Режим доступа: http://citforum.ru/operating_systems/articles/posix.shtml. – Загл. с экрана.
6. ИНТУИТ. Национальный открытый университет [Электронны ресурс] // В. Галатенко, Программирование в стандарте POSIX : сайт. – Электрон. текстов. дан. – [Б. м.], 2015. – Режим доступа: http://www.intuit.ru/studies/courses/47/47/lecture/1397. – Загл. с экрана.
7. GNU Operating System [Электронный ресурс] // GNU Coding Standards : сайт. – Электрон. текстов. дан. – [Б. м.], 2015. – Режим доступа: http://www.gnu.org/prep/standards. – Загл. с экрана.
8. Роббинс А. Linux: программирование в примерах [Текст] : учеб. метод. пособие / А. Роббинс; пер. с англ. – М. : КУДИЦ-ОБРАЗ, 2005. – 656 с.
9. Белорусов, А. Перспективы развития мирового рынка высоких технологий [Текст] / А. Белорусов, В. Вовченко // Белорусский журнал международного права и международных отношений. – 2002. – № 2. – С. 80–85.
10. Журнал LinuxFormat [Электронный ресурс] // Правильные аргументы : сайт. – Электрон. текстов. дан. – [Б. м.], 2015. – Режим доступа: http://wiki.linuxformat.ru/wiki/LXF112:Getopt. – Загл. с экрана.
11. Heap AltLinux [Электронный ресурс] // Ввод, вывод и конвейер: сайт. – Электрон. текстов. дан. – [Б. м.], 2008. – Режим доступа: http://heap.altlinux.org/modules/linux_pipeline/index.html. – Загл. с экрана.
12. OpenNET [Электронный ресурс] // Advanced Bash-Scripting Guide – Электрон. текстов. дан. – [Б. м.], 2016. – Режим доступа: http://www.opennet.ru/docs/RUS/bash_scripting_guide/. – Загл. с экрана.