

[Новые вопросы](#) > [с](#)

Переменная среды Linux IFS не используется функцией библиотеки gcc wordexp Posix C для разделения слов

Среда

ОС: Ubuntu 20.4, Centos 8, macOS Catalina 10.15.7

Язык: C, C ++

Компилятор: gcc (самые последние версии для каждой ОС)

Проблема

Я использую функцию библиотеки Posix wordexp , чтобы получить расширение строк в виде оболочки.

Расширение работает нормально с одним исключением: когда я устанавливаю переменную среды \$ IFS на что-то другое, кроме пробелов, например ':', похоже, это не влияет на разделение слов, которое продолжает выполняться только на пробелах, независимо от значения IFS. .

bash тест

Страница руководства для wordexp для Linux <https://B/man7.org/linux/man-pages/man3/wordexp.3.html> говорится:

«Функция wordexp () выполняет расширение строки ...»

«Разделение полей выполняется с использованием переменной среды \$ IFS. Если она не задана, разделителями полей являются пробел, табуляция и новая строка».

Вот почему я ожидал, что wordexp будет вести себя в этом отношении так же, как bash .

Во всех перечисленных ОС я получил точно такой же точно правильный и ожидаемый результат при изменении набора символов, используемого для разделения:

По умолчанию (IFS не установлен)

```
read -a words <<<"1 2:3 4:5"
for word in "${words[@]}; do echo "$word"; done
```

Правильно распределяется по пространству и дает результат:

Похожие вопросы

- 6 [Статическая ссылка на функцию разделяемой библиотеки в gcc](#)
- 5 [Как создать простой make-файл для gcc в Linux?](#)
- 3 [Как скомпилировать статическую библиотеку в Linux?](#)
- 7 [Допустимые символы в именах переменных среды Linux](#)
- 6 [Clang vs GCC для моего проекта разработки Linux](#)

```
1
2:3
4:5
```

При установке IFS в ':'

```
IFS=':' read -a words <<<"1 2:3 4:5"
for word in "${words[@]}"; do echo "$word"; done
```

Правильно разбивается на ':' и дает результат:

```
1 2
3 4
5
```

Тест кода C

Но выполнение приведенного ниже кода дает тот же результат независимо от того, установлена ли переменная среды IFS или нет:

Код C:

```
#include <stdio.h>
#include <wordexp.h>
#include <stdlib.h>


static void expand(char const *title, char const *str)
{
    printf("%s input: %s\n", title, str);
    wordexp_t exp;
    int rcode = 0;
    if ((rcode = wordexp(str, &exp, WRDE_NOCMD)) == 0) {
        printf("output:\n");
        for (size_t i = 0; i < exp.we_wordc; i++)
            printf("%s\n", exp.we_wordv[i]);
        wordfree(&exp);
    } else {
        printf("expand failed %d\n", rcode);
    }
}
```

6 Как сохранить переменные среды при использовании sudo

5 Почему препроцессор C интерпретирует слово «linux» как константу «1»?

2 Есть ли функция для перебора переменных среды POSIX?

2 кто может объяснить мне, почему эта команда atoi работает и как

 Получи \$100 на хостинг на 60 дней.
Регистрируйся!

Новые вопросы

1 Перечислено {a} e = 1; действительный?

2 Почему он дает другой адрес с той же переменной?

2 непреднамеренный выходной символ p при переворачивании строки ДНК в C

2 Почему эта операция сдвига плюс побитовая работает только до 31?

2 Доступ к значению переменной без объявления его в прототипе функции

2 Самый простой способ проверить выбор, сделанный пользователем

4 if-else в do-while в c

2 Почему расширение знака ff для целого числа без знака?

2 Почему возникает ошибка сегментации?

```

int main()
{
    char const *str = "1 2:3 4:5";

    expand("No IFS", str);

    int rcode = setenv("IFS", ":", 1);
    if ( rcode != 0 ) {
        perror("setenv IFS failed: ");
        return 1;
    }

    expand("IFS=':'", str);

    return 0;
}

```

Результат во всех ОС одинаковый:

```

No IFS input: 1 2:3 4:5
output:
1
2:3
4:5
IFS=':' input: 1 2:3 4:5
output:
1
2:3
4:5

```

В качестве примечания, приведенный выше фрагмент был создан для этого сообщения – я провел тестирование с более сложным кодом, который подтвердил, что переменная среды действительно установлена правильно.

Обзор исходного кода

Я просмотрел исходный код реализации функции wordexp , доступный по адресу

<https://code.woboq.org/userspace/glibc/posix/wordexp.c.html>, и похоже, что он использует \$IFS, но, возможно, непоследовательно или, возможно, это ошибка.

В частности:

В теле wordexp , которое начинается в строке 2229 , он получает значение переменной среды

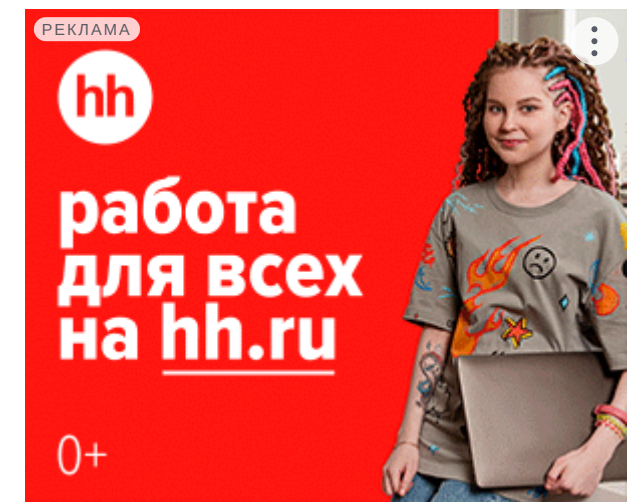
2 Возможный недостаток «Целочисленное переполнение или перенос» в этом коде C

[Все вопросы по теме c >](#)

C

C – это язык программирования общего назначения, используемый для системного программирования (ОС и встраиваемых), библиотек, игр и кроссплатформенности. Этот тег следует использовать с общими вопросами, касающимися языка C, как это определено в стандарте ISO 9899 (последняя версия 9899: 2018, если не указано иное, а также для запросов, специфичных для версии, с c89, c99, c11 и т. Д.). C отличается от C++ и не должен сочетаться с тэгом C++ без разумной причины.

[Подробнее про c...](#)



IFS и обрабатывает его:
строки 2273 – 2276:

```
/* Find out what the field separators are.  
 * There are two types: whitespace and non-whitespace.  
 */  
ifs = getenv ("IFS");
```

Но позже в функции, похоже, не используйте значения \$ IFS для разделения слов.
Это похоже на ошибку, если только "разделители полей" в строке 2273 и "разделитель слов" в строке 2396 означают разные вещи.
строки 2395 – 2398:

```
default:  
/* Is it a word separator? */  
if (strchr (" \t", words[words_offset]) == NULL)  
{
```

Но в любом случае код, похоже, использует только пробел или табуляцию в качестве разделителя. в отличие от bash , который учитывает значения разделителя набора IFS.

Вопросы

Я что-то упустил и есть способ заставить wordexp разделить символы, отличные от пробелов?
Если разделение выполняется только по пробелам, это ошибка в
gcc реализация библиотеки или
на странице руководства Linux для wordexp , где утверждается, что \$ IFS можно использовать
для определения разделителей
Заранее благодарим за все ваши комментарии и идеи!

Сводка ответов и обходной путь

В принятом ответе был намек на то, как добиться разделения на непробельные символы из \$ IFS:
вам нужно установить \$ IFS и поместить строку, которую вы хотите разделить, как значение для временной переменной среды, а затем вызвать wordexp для этой временной переменной. Это показано в обновленном коде ниже.

Хотя такое поведение, которое видно в исходном коде, на самом деле не может быть ошибкой, мне определенно кажется сомнительным дизайнерским решением...

Обновленный код:

```
#include <stdio.h>
#include <wordexp.h>
#include <stdlib.h>

static void expand(char const *title, char const *str)
{
    printf("%s input: %s\n", title, str);
    wordexp_t exp;
    int rcode = 0;
    if ((rcode = wordexp(str, &exp, WRDE_NOCMD)) == 0) {
        printf("output:\n");
        for (size_t i = 0; i < exp.we_wordc; i++)
            printf("%s\n", exp.we_wordv[i]);
        wordfree(&exp);
    } else {
        printf("expand failed %d\n", rcode);
    }
}

int main()
{
    char const *str = "1 2:3 4:5";

    expand("No IFS", str);

    int rcode = setenv("IFS", ":", 1);
    if ( rcode != 0 ) {
        perror("setenv IFS failed: ");
        return 1;
    }

    expand("IFS=':'", str);

    rcode = setenv("FAKE", str, 1);
    if ( rcode != 0 ) {
        perror("setenv FAKE failed: ");
        return 2;
    }

    expand("FAKE='FAKE'", str);
}
```

Что дает результат:

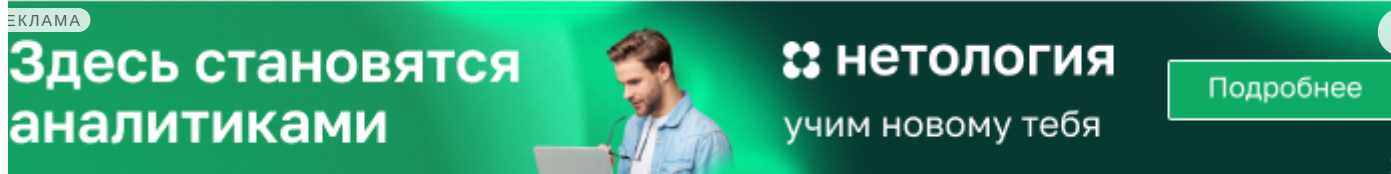
```
No IFS input: 1 2:3 4:5
output:
```

```
1
2:3
4:5
IFS=':' input: 1 2:3 4:5
output:
1
2:3
4:5
FAKE input: ${FAKE}
output:
1 2
3 4
5
```

c linux gcc posix ifs

4

Leo 21 Янв 2021 в 09:30



2 ответа

Лучший ответ

Вы сравниваете яблоки с апельсинами. `wordexp()` разбивает строку на отдельные токены так же, как это делает оболочка. Встроенная оболочка `read` не следует тому же алгоритму; он просто разделяет слова. Вы должны сравнивать `wordexp()` с тем, как анализируются аргументы скрипта или функции оболочки:

```
#!/bin/sh

printwords() {
    for arg in "$@"; do
```

```
    printf "%s\n" "$arg"
done
}

echo "No IFS input: 1 2:3 4:5"
printwords 1 2:3 4:5
echo "IFS=':' input: 1 2:3 4:5"
IFS=:
printwords 1 2:3 4:5
```

Это производит

```
No IFS input: 1 2:3 4:5
1
2:3
4:5
IFS=':' input: 1 2:3 4:5
1
2:3
4:5
```

Точно так же, как программа C.

Теперь самое интересное. Я не смог найти это явно упомянутое как таковое в документации POSIX при быстром сканировании, но [bash manual](#) говорит о разбиении слов:

Обратите внимание, что если расширения не происходит, разделение не выполняется.

Давайте попробуем версию, которая расширяет параметры в своих аргументах:

```
#!/bin/sh

printwords() {
    for arg in "$@"; do
        printf "%s\n" "$arg"
    done
}

foo=2:3
printf "foo = %s\n" "$foo"
printf "No IFS input: 1 \ $foo 4:5\n"
```

```
printwords 1 $foo 4:5
printf "IFS=':' input: 1 \"$foo 4:5\\n\"
IFS=:
printwords 1 $foo 4:5
```

Которые при запуске через оболочки, такие как dash , ksh93 или bash (но не zsh , если вы не включите параметр SH_WORD_SPLIT) , производит

```
foo = 2:3
No IFS input: 1 $foo 4:5
1
2:3
4:5
IFS=':' input: 1 $foo 4:5
1
2
3
4:5
```

Как видите, аргумент, имеющий параметр, подвергался разделению полей, но не буквально. Внесение того же изменения в строку в вашей программе на C и запуск `foo=2:3 ./wordexp` выводит на печать то же самое.

2

Shawn 21 Янв 2021 в 16:50

Давайте наивно предположим, что POSIX понятен, и попробуем с ним поработать. Возьмем [wordexp\(\) из posix](#):

Аргумент слов – это указатель на строку, содержащую одно или несколько расширяемых слов. Расширения должны быть такими же, как если бы они были выполнены интерпретатором командной строки, если бы слова были частью командной строки, представляющей аргументы для утилиты. [...]

Итак, перейдем к «интерпретатору командной строки». Из [командного языка оболочки posix](#):

2.1 Введение в оболочку

[...]

Оболочка разбивает ввод на токены: слова и операторы; см. Распознавание токенов.

[.....]

2.3 Распознавание токенов

[...]

Если текущий символ – это <пробел> без кавычек, любой токен, содержащий предыдущий символ, разделяется, и текущий символ должен быть отброшен.

Если предыдущий символ был частью слова, текущий символ должен быть добавлен к этому слову.

[...]

В основном здесь применяются все разделы 2.3 Token Recognition – это то, что делает `wordexp()` – распознавание токенов плюс некоторые расширения. А также самое важное о [разделении полей](#), акцент мой :

После расширения параметра (расширение параметра), подстановки команд (подстановка команд) и арифметического раскрытия (арифметическое расширение) оболочка просканирует результаты расширений и замен , которые не были заключены в двойные кавычки для поля может произойти разделение и несколько полей.

IFS влияет на разделение полей, это влияет на то, как результат других расширений выражается словами. IFS не влияет на то, как строка разбивается на токены, она по-прежнему разделяется с использованием <blank> – табуляции или пробела. Итак, поведение, которое вы видите.

Другими словами, когда вы вводите `IFS=:` в терминале, вы не начинаете разделять токены на IFS , как `echo:Hello:World` , но продолжаете разделять части команд с помощью пробелов.

В любом случае, страница руководства верна ...: p

Я что-то упустил, и есть ли способ разбить `wordxp` на символы, отличные от пробелов?

Нет. Если вы хотите использовать пробелы в словах, заключите аргументы в кавычки, как в оболочке. `"a b" "c d" "e"` .

Если разделение выполняется только по пробелам, это ошибка в

Нет: p

2

KamilCuk 21 Янв 2021 в 15:47

Языки		Библиотеки		Фреймворки	Мобайл
JavaScript	Java	Pandas	React native	Django	Android
HTML	C	React	TensorFlow	Laravel	iOS
CSS	C#	jQuery	Matplotlib	Selenium	Android Studio
PHP	C++	Angular	Bootstrap	Flask	Dart
Python	R	Numpy	Keras	Flutter	Ionic
SQL	Swift	Vue.js	OpenCV	Docker	Apache Cordova

 En Español

Самоучитель Python

