16. Сопоставления с образцом

Библиотека GNU C обеспечивает средства сопоставления с образцом для двух видов шаблонов: регулярные выражения и универсальные символы имени файла.

16.1 Универсальное сопоставление

Этот раздел описывает, как шаблон универсальных символов соответствует специфической строке. Результат - ответ да или никакого ответа: строка удовлетворяет шаблону или нет. Символы, описанные здесь объявлены в " fnmatch.h ".

```
int fnmatch (const char *pattern, const char *string, int flags)
```

Эта функция проверяет, соответствует ли указанная строка шаблону. Она возвращает 0, если они соответствуют; иначе, она возвращает значение FNM_NOMATCH отличное от нуля.

Flags - комбинация флаговых битов, которые изменяют подробности соответствия. См. ниже список определенных флагов.

В Библиотеке GNU C, fnmatch не может испытывать "ошибку", она всегда возвращает ответ, преуспевает ли соответствие. Однако, другие реализации fnmatch могут иногда сообщать "ошибки", возвращая значения отличные от нуля, которые не равны FNM_NOMATCH.

Вот доступные флаги для аргумента flags:

```
FNM_FILE_NAME
```

Если этот флаг установлен, универсальные конструкции символов в шаблоне не могут соответствовать " / " в строке. Таким образом, единственый способ соответствовать " / " явно указать " / " в шаблоне.

FNM_PATHNAME

Это – побочный результат исследования для FNM_FILE_NAME; он исходит ИЗ POSIX.2. Мы не рекомендуем это имя, потому что мы не используем термин "имя пути" для имен файла.

FNM_PERIOD

Обрабатывает "." особенно, если она появляется в начале строки. Если этот флаг установлен, универсальные конструкции символов в шаблоне не могут соответствовать "." (первый символ строки).

Если Вы устанавливаете, и FNM_PERIOD и FNM_FILE_NAME, то "." после "/" трактуется также как к "." в начале строки. (Оболочка использует FNM_PERIOD и FNM_FILE_NAME флаги вместе для соответствия имен файлов.)

FNM_NOESCAPE

Не обрабатывает символ `\' в шаблонах. Обычно, " \ " означает (цитирует) следующий символ непосредственно. Когда цитирование допускается, шаблон " \? " означает строку "? ", потому что вопросительный знак в шаблоне действует подобно обычному символу.

Если Вы используете FNM_NOESCAPE, то " \ " является обычным символом.

FNM_LEADING_DIR

Игнорирует конечную последовательность символов, начинающихся с " / " в строке.

Если этот флаг установлен, и " foo* " и " foobar " как шаблоны, соответствуют строке " foobar/frobozz ".

FNM_CASEFOLD

Игнорирует case при сравнении строки и шаблона.

16.2 Globbing

Типичное использование групповых символов – для соответствия файлов в каталоге, и создании списка всех соответствий. Это называется globbing.

Вы могли бы делать это используя fnmatch, читая входы каталога один за другим и проверяя каждый с fnmatch. Но это было бы медлено (и сложно, так как Вы будете должны обработать подкаталоги вручную).

Библиотека обеспечивает функцию glob, чтобы делать это с использованием удобных универсальных символов. Glob и другие символы в этом разделе объявлены в " glob.h ".

Вызов glob

Результат globbing - вектор имен файлов. Чтобы возвращать этот вектор, glob использует специальный тип данных, glob _t, который является структурой. Вы передаете glob адрес структуры, и она вносит поля структуры, чтобы сообщить Вам результаты.

```
glob_t                (тип данных)
```

Этот тип данных содержит указатель на вектор слов. Более точно, он содержит, и адрес вектора слов и размер.

```
Gl_pathc Число элементов в векторе.
Gl_pathv Адрес вектора. Это поле имеет тип char **.
Gl_offs Смещение первого реального элемента вектора, от
        номинального адреса в gl_pathv поле. В отличие от других полей,
        это -- всегда ввод glob, а не вывод из нее (т. е. вы должны
        указать его).
```

Если Вы используете смещение отличное от нуля, то много элементов в начале вектора будут оставлены пустыми. (Функция glob заполняет их пустыми указателями.)

Gl_offs поле значимо только, если Вы используете GLOB_DOOFFS флаг.

Иначе, смещение - всегда нуль независимо от того, что находится в этом поле, и первый реальный элемент расположен в начале вектора.

```
int glob (const char *pattern, int flags, int (*errfunc) (const char *filename, int error-code), glob_t *vector_ptr)
```

Эта функция делает globbing, используя указанный шаблон в текущем каталоге. Она помещает результат в недавно размещенном векторе, и сохраняет размер и адрес этого вектора в *vector_ptr. Flags аргумент - комбинация битовых флагов; см. Раздел 16.2.2 [Флаги для Globbing].

Результат globbing - последовательность имен файлов. Glob резервирует строку для каждого возникающего в результате слова, и вектор типа char **, чтобы сохранить адреса этих строк. Последний элемент вектора - пустой указатель. Этот вектор называется вектором слов.

Чтобы возвратить этот вектор, glob сохраняет и адрес и длину (число элементов, не считая завершающий пустой символа) в *vector_ptr.

Обычно, glob сортирует имена файлов в алфавитном порядке перед их возвращением. Вы можете указать флаг GLOB_NOSORT, если Вы хотите получать информацию, с наибольшей скоростью.

Если glob преуспевает, она возвращает 0. Иначе, она возвращает один из этих кодов ошибки:

GLOB_ABORTED Имелась ошибка открытия каталога, и Вы использовали флаг GLOB_ERR, или ваша заданная errfunc возвратила значение отличное от нуля. См. ниже объяснение GLOB_ERR и errfunc.

GLOB_NOMATCH Шаблон не соответствовал ни каким существующим файлам. Если Вы используете GLOB_NOCHECK флаг, то Вы, никогда не получаете этот код ошибки, потому что этот флаг сообщает, чтобы glob симулировал что шаблону соответствует по крайней мере один файл.

GLOB_NOSPACE Было невозможно зарезервировать память, чтобы содержать результат.

В случае ошибки, glob сохраняет информацию в *vector_ptr относительно всей соответствий, которые она уже нашла.

Флаги для Glob

Этот раздел описывает флаги, которе Вы можете определять в аргументе flags в glob. Выберите флаги которые Вы хотите, и объедите их оператором OR (| в Си).

GLOB_APPEND

Добавлять слова от этого поиска к вектору слов, произведенных предыдущими обращениями к glob.

Для этого, Вы не должны изменить содержимое структуры вектора слов между обращениями к glob. И, если Вы устанавливаете GLOB_DOOFFS в первом обращении к glob, Вы должны также установить его, когда Вы добавляете.

Обратите внимание, что указатель, сохраненный в gl_pathv может больше не быть допустимым после того, как Вы вызываете glob второй раз, потому что glob может переместить вектор. Так что всегда берите gl_pathv из структуры _t glob после каждого обращения к glob; никогда не сохраните указатель между обращений.

GLOB_DOOFFS

Оставьте пустые места в начале вектора слов. Gl_offs поле говорит сколько мест оставить. Пустые места содержат пустые указатели.

GLOB_ERR

Сразу же сообщает ошибку, если имеется любая трудность. Такие трудности могут включать каталог, в котором Вы не имеете необходимого доступа. Обычно, glob пробует продолжить несмотря на любые ошибки, и читает любые каталоги, какие может.

Вы можете осуществлять управление, определяя функцию обработчика ошибки errfunc, когда Вы вызываете glob. Если errfunc - не пустой указатель, то glob не отказывается сразу же, когда она не может читать каталог; взамен, она вызывает errfunc с двумя аргументами, примерно так:

(*errfunc) (filename, error-code)

Filename - имя каталога, который glob не может открыть, или не может читать, а error-code - значение errno, которое было сообщено glob.

Если функция обработчика ошибки возвращает не нуль, то glob завершается сразу же. Иначе, она продолжается.

GLOB_MARK Если шаблон соответствует имени каталога, конкатенирует " / " к имени каталога при его возвращении.

GLOB_NOCHECK Если шаблон не соответствует ни каким именам файлов, возвращает шаблон непосредственно, как будто это было имя файла. (Обычно, когда шаблон не соответствует чему - нибудь, glob говорит что не имелось никакого соответствия.)

GLOB_NOSOR(***) Не сортирует имена файлов. (Практически, порядок будет зависеть от порядка входов в каталоге.) Единственая причина не сортировать состоит в том, чтобы сохранить время.

GLOB_NOESCAPE Не обрабатывает символ ` \ ' в шаблонах. Обычно, " \ " цитирует следующий символ, выключая специальное значение так, чтобы он соответствовал только непосредственно символу. Когда цитирование допускается, шаблон " \? " соответствует только строке "? ", потому что вопросительный знак в шаблоне действует подобно обычному символу.

Если Вы используете GLOB_NOESCAPE, то " \ " является обычным символом.

Glob делает это, вызывая функцию fnmatch. Она обрабатывает флаг GLOB_NOESCAPE, включая FNM_NOESCAPE флаг в обращениях к fnmatch.

16.3 Соответствия Регулярных Выражений

Библиотека GNU C поддерживает два интерфейса для соответствия регулярных выражений. Один - стандартный POSIX.2 интерфейс, а другой - тот, который система GNU использовала много лет.

Оба интерфейса объявлены в заглавном файле " regex.h ". Если Вы определяете _POSIX_C_SOURCE, то будут объявлены только POSIX.2 функции, структуры, и константы.

POSIX Регулярные Выражения

Прежде, чем Вы можете фактически использовать регулярное выражение, Вы должны откомпилировать его. Это – не истинная трансляция, это производит специальную структуру данных, а не машинные команды. Но это – подобно обычной трансляции, цель которой дать Вам возможность " выполнить" шаблон быстро. (См. Раздел 16.3.3 [Соответствие POSIX Регулярным Выражениям], для того, как использовать компилируемое регулярное выражение для соответствия.)

Имеется специальный тип данных для компилируемых регулярных выражений:

```
regex_t              (тип данных)
```

Этот тип содержит компилируемое регулярное выражение. Это – фактически структура. Она имеет только одно поле, которое ваши программы должны рассмотреть: re_nsub . Это поле содержит некоторое число вводных подвыражений регулярного выражения.

Имеются и другие поля, но мы не описываем их здесь, потому что только функции в библиотеке должны использовать их.

После того, как Вы создаете объект regex_t, Вы можете компилировать регулярное выражение в ней, вызывая regcomp.

```
int regcomp (regex_t *compiled, const char *pattern, int cflags)
```

Функция regcomp "компилирует" регулярное выражение в структуру данных, которую Вы можете использовать с regexec, чтобы искать соответствующие строки. Компилируемый формат регулярного выражения разработан для эффективного соответствия. Regcomp сохраняет его в *compiled.

Вам нужно только зарезервировать объект типа regex_t и передать адрес regcomp.

Аргумент cflags допускает Вам, определять различные опции, которые управляют синтаксисом и семантикой регулярных выражений.

Если Вы используете флаг REG_NOSUB, то regcomp, опускает из компилируемого регулярного выражения информацию, необходимую для записи соответствий подвыражений. В этом случае, Вы можете также указывать 0 для matchptr и nmatch аргументов, когда Вы вызываете regexec.

Если Вы не используете REG_NOSUB, то компилируемое регулярное выражение имеет запись соответствия подвыражений. Также, regcomp сообщает Вам, сколько подвыражений имеет шаблон, сохраняя число в compiled->re_nsub. Вы можете использовать это значение, чтобы решить, какую длину массива зарезервировать, чтобы содержать информацию относительно соответствий подвыражений.

Regcomp возвращает 0, если она преуспевает в компилировании регулярного выражения; иначе, она возвращает код ошибки отличный от нуля (см. таблицу ниже). Вы можете использовать regerror, чтобы произвести строку сообщения об ошибках для значений отличных от нуля.

Имеются возможные значения отличные от нуля, которые regcomp может возвращать:

REG_BADBR Имеется недопустимая конструкция " \{. . .\} " в регулярном выражении. Допустимая " \{. . .\} " конструкция должна содержать или одиночное число, или два числа в увеличивающемся порядке, отделенные запятой.

REG_BADPAT Имелась синтаксическая ошибка в регулярном выражении.

REG_BADRPT Оператор повторения типа " ? " или " * " оказался, в плохой позиции (без предшествующего подвыражения).

REG_ECOLLATE Регулярное выражение сносится на недопустимый элемент объединения (не определенный в текущем стандарте для строкового объединения). См. Раздел 19.3 [Категории Стандарта].

REG_ECTYPE Регулярное выражение ссылается на недопустимое символьное имя класса.

REG_EESCAPE Регулярное выражение закончено " \ ".

REG_ESUBREG Имелось недопустимое число в " \digit " конструкции.

REG_EBRACK Имелись несбалансированные квадратные скобки в регулярном выражении.

REG_EPAREN Расширенное регулярное выражение имело незакрытые скобки, или базисное регулярное выражение имело несбалансированные "\(" и "\)".

REG_EBRACE Регулярное выражение имело несбалансированные "\{" и "\}".

REG_ERANGE Одна из оконечных точек в выражении диапазона была недопустима.

REG_ESPACE Regcomp не хватает памяти.

Флаги для POSIX Регулярных Выражений

Это – битовые флаги, который Вы можете использовать в cflags операнде при компилировании регулярного выражения с regcomp.

REG_EXTENDED

Обрабатывает шаблон как расширенное регулярное выражение, а не как базисное регулярное выражение.

REG_ICASE

Игнорирует case при соответствии символов.

REG_NOSUB

Не сохраняет содержимое matches_ptr массива.

REG_NEWLINE

Обрабатывает символ перевода строки в строке как деление строки на многократные строки, так, чтобы " $ " мог соответствовать перед символом перевода строки, а " ^ " мог соответствовать после. Также, не разрешает "." соответствовать символу перевода строки, и не разрешает " [^. . .] " соответствовать символу перевода строки.

Иначе, символ перевода строки действует подобно любому другому обычному символу.

Соответствие Компилируемого POSIX Регулярного Выражения

Если только Вы компилировали регулярное выражение, как описано в Разделе 16.3.1 [Трансляция POSIX Регулярных выражений], Вы можете применять его к строкам используя regexec. Соответствие где-нибудь внутри строки считается как успех, если регулярное выражение не содержит символы " ^ " или " $ ".

int regexec (regex_t *compiled, char *string, size_t nmatch, regmatch_t matchptr [], int eflags) (функция)

Эта функция пробует подобрать соответствие компилируемому регулярному выражению *compiled.

Regexec возвращает 0 если tcnm соответствие выражению; иначе, она возвращает значение отличное от нуля. См. таблицу ниже для того, что означают значения отличные от нуля. Вы можете использовать regerror, чтобы произвести строку сообщения об ошибках для значений отличных от нуля.

Аргумент eflags – слово битовых флагов, которые дают возможность различным опциям.

Если Вы хотите получать информацию относительно части строки которая фактически соответствовала регулярному выражению или подвыражению, используйте аргументы matchptr и nmatch. Иначе, укажите 0 для nmatch, и пустой указатель для matchptr.

Функция regexec принимает следующие флаги в аргументе eflags:

REG_NOTBOL Не расценивает начало заданной строки как начало строки; более вообще, не делает ни каких предположений относительно того, что текст мог бы предшествовать ей.

REG_NOTEOL Не расценивает конец заданной строки как конец строки; более обще, не делает ни каких предположений относительно того, что текст мог бы следовать за ней.

Имеются возможные значения отличные от нуля, которые regexec может возвращать:

REG_NOMATCH

Шаблон не соответствовал строке. Это в общем не ошибка.

REG_ESPACE Regexec не хватило памяти.

Результаты Соответствия с Подвыражениями

Когда regexec находит соответствия подвыражениям шаблона, она записывает, которым частям строки они соответствуют. Она возвращает эту информацию, сохраняя смещения в массиве, чьи элементы являются структурами типа regmatch_t. Первый элемент массива (индекс 0) записывает часть строки, которая соответствовала всему регулярному выражению. Каждый другой элемент массива записывает начало и конец части, которая соответствовала одиночному вводному подвыражению.

regmatch_t

Это - тип данных matcharray массива, который Вы передаете к regexec. Он содержит два поля-структуры, следующим образом: Rm_so - Смещение начала подстроки в строке. Добавьте это значение к строке, чтобы получить адрес этой части. Rm_eo - Смещение конца подстроки в строке.

regoff_t

Regoff_t - побочный результат исследования для другого целого типа со знаком. Поля regmatch_t имеют тип regoff_t. Regmatch_t элементы соответствуют подвыражениям позиционно; первый элемент (индекс 1) хранит, где первое согласованное подвыражение, второй элемент записывает второе подвыражение, и так далее. Порядок подвыражений - порядок, в котором они начинаются.

Когда Вы вызываете regexec, Вы определяете длину matchptr массива, с nmatch аргументом.

Это сообщает regexec сколько элементов сохранить. Если фактическое регулярное выражение имеет больше чем nmatch подвыражений, то, Вы не будет получать информацию о смещениях относительно остальной их части.

Еслио Вы не хотите, чтобы regexec возвращал любую информацию относительно подвыражений, Вы можете обеспечить 0 для nmatch, или использовать флаг REG_NOSUB, когда Вы компилируете шаблон с regcomp.

Осложнения в Соответствиях Подвыражений

Иногда подвыражение соответствует подстроке без символов. Это случается, когда " f\(o*\) " соответствует строке " fum ". (Оно действительно соответствует только " f ".) В этом случае, оба смещения идентифицируют отметку в строке, где была найдена пустая подстрока. В этом примере, оба смещения 1.

Иногда все регулярное выражение может соответствовать без использования некоторых из подвыражений вообще, например, когда " ba\(na\)* " соответствует строке " ba ", вводное подвыражение не используется. Когда это случается, regexec сохраняет -1 в обоих полях элемента для этого подвыражения.

Иногда при соответствии всего регулярного выражения некоторая подстрока может соответствовать специфическому подвыражению больше чем один раз например, когда " ba\(na\)* " соответствует строка " bananana ", вводное подвыражение соответствует три раза. Когда это случается, regexec обычно сохраняет смещения последней части строки, которая соответствовала подвыражению. В случае " bananana ", эти смещения - 6 и 8.

Очистка POSIX Regexp Соответствий

Когда Вы закончили использование компилируемого регулярного выражения, Вы можете освободить память, которую оно использует, вызывая regfree.

```
void regfree (regex_t *compiled)  (функция)
```

Вызов regfree освобождает всю память, на которую *compiled указывает. Включая различные внутренние поля структуры regex_t, которые не описаны в этом руководстве. Regfree не освобождает объект *compiled непосредственно.

Вы должны всегда освобождать место в структуре regex_t с regfree перед использованием структуры, чтобы компилировать другое регулярное выражение.

Когда regcomp или regexec сообщает об ошибке, Вы можете использовать функцию regerror, преобразовать ее в строку сообщения об ошибках.

```
size_t regerror (int errcode, regex_t *compiled, char *buffer, size_t length) (функция)
```

Эта функция производит строку сообщения об ошибках для кода ошибки errcode, и сохраняет строку в length байтах памяти, начинающейся с buffer. Для аргумента compiled, обеспечьте то же самое регулярное выражение, с которой работал regcomp или regexec когда получил ошибку. В качестве альтернативы, Вы можете обеспечивать пустой указатель для compiled; Вы будете все еще получать значимое сообщение об ошибках, но оно может не быть детализировано.

Если сообщение об ошибках не помещается в length байтах (включая пустой символ завершения), то regerror усекает его. Эта строка всегда с нулевым символом в конце даже если она была усечена.

Возвращаемое значение regerror - минимальный length, для сохранения всего сообщения об ошибках. Если он меньше чем length, то сообщение об ошибках не было усечено, и Вы можете использовать его. Иначе, Вы должны вызвать regerror снова с большим buffer.

Вот функция, которая использует regerror, но всегда динамически, резервируя буфер для сообщения об ошибках:

```
char *get_regerror (int errcode, regex_t *compiled)
{
  size_t length = regerror(errcode,compiled,NULL,0);
  char *buffer = xmalloc (length);
  (void) regerror (errcode, compiled, buffer, length);
  return buffer;
}
```

16.4 Разложение Слов в стиле оболочки

Разложение Слов означает процесс разбивания строки в слова и замену переменных, команд, и универсальных символов, точно так как делает оболочка.

Например, когда Вы пишите " ls -l foo.c ", эта строка разбивается в три отдельных слова " ls ", " -l " и " foo.c ". Это базисная функция разложения слов.

Когда Вы пишите " ls *.c ", это может стать многими словами, потому что слово " *.c " может быть заменено на любое число имен файлов. Это называется разложением универсального символа, и это также часть разложения слова.

Когда Вы используете " ECHO $PATH " чтобы печатать ваш путь, Вы пользуетесь преимуществом замены переменной, которая является также частью разложения слова.

Обычные программы могут выполнять разложение слова точно так же как оболочка, вызывая библиотечную функцию wordexp.

Стадии Разложения Слова

Когда разложение слова применяется к последовательности слов, выполняются следующие преобразования в порядке, показанном здесь:

1. Разложение Тильды: Замена " ~foo " на имя исходного (home) каталога " foo ".
2. Затем, применяются три различных преобразования в том же самом шаге, слева направо:
    - Замена переменных: Переменные среды заменяются для ссылок типа " $foo ".
    - Замена Команд: Конструкции типа "" cat foo "" и эквивалент " $(cat foo) " заменены на вывод внутренней команды.
    - Арифметическое разложение: Конструкции типа " $(($x-1)) " заменены на результат арифметического вычисления.
3. Разбивание Поля: разбиение текста в слова.
4. Разложение Универсальных символов: замена конструкции типа " *.c " на список " .c " имен файлов. Разложение Универсального символа применяется к всему слову одновременно, и заменяет это слово на 0 или большое количество имен файлов, которые являются самостоятельными словами.
5. Удаление Кавычек: стирание кавычек, теперь, когда они сделали их работу, запрещая вышеупомянутые преобразования когда нужно.

Для подробностей этих преобразований, и как написать использующие их конструкции, см. Руководство BUSH (должно появиться).

Вызов wordexp

Все функции, константы и типы данных для разложения слова объявлены в заглавном файле " wordexp.h ".

Разложение Слова производит вектор слов (строк). Чтобы возвращать этот вектор, wordexp использует специальный тип данных, wordexp_t, который является структурой. Вы передаете wordexp адрес структуры, и она вносит поля структуры, чтобы сообщить Вам результаты.

```
wordexp_t
```

Этот тип данных содержит указатель на вектор слов. Более точно, в нем записаны, и адрес вектора слов и размер. We_wordc - Число элементов в векторе. We_wordv - Адрес вектора. Это поле имеет тип char **. We_offs - Смещение первого реального элемента вектора от номинального адреса в we_wordv поле. В отличие от других полей, это всегда ввод к wordexp, а не вывод из нее.

Если Вы используете смещение отличное от нуля, то многие элементы в начале вектора будут оставлены пустыми. (Wordexp функция заполняет их пустыми указателями.) We_offs поле значимо только, если Вы используете WRDE_DOOFFS флаг. Иначе, смещение - всегда нуль независимо от того, что находится в этом поле, и первый реальный элемент находится в начале вектора.

```
int wordexp (const char *words, wordexp_t *word-vector-ptr, int flags)
```

Выполните разложение слова на строке слов, помещая результат в недавно размещенном векторе, и сохраните размер и адрес этого вектора в *word-vector-ptr. Аргумент flags – комбинация битовых флагов; см. Раздел 16.4.3 [Флаги для Wordexp].

Вы не должны использовать любой из символов " | & ; < > " в строке слов, если они не заключены в кавычки; аналогично для символа перевода строки. Если Вы используете эти символы без кавычек, Вы получите WRDE_BADCHAR код ошибки. Не используйте круглые скобки или фигурные скобки, если они заключены в кавычки или часть конструкции разложения слова. Если Вы используете кавычки " ' `, они должены войти попарно.

Результаты разложения слов – последовательность слов. Функция wordexp зарезервирует строку для каждого возникающего в результате слова, и вектор типа char **, чтобы сохранить адреса этих строк. Последний элемент вектора – пустой указатель. Этот вектор называется вектором слов.

Чтобы возвращать этот вектор, wordexp сохраняет, и адрес и длину (число элементов, не считая завершающий пустой символ) в *word-vector-ptr.

Если wordexp завершает работу успешно, она возвращает 0. Иначе, она возвращает один из этих кодов ошибки:

WRDE_BADCHAR

Входные строковые слова содержат незащищенный кавычками недопустимый символ типа " | ".

WRDE_BADVAL

Входная строка обращается к неопределенной переменной оболочки, и Вы использовали флаг WRDE_UNDEF, чтобы запретить такие ссылки.

WRDE_CMDSUB

Входная строка использует замену команды, и Вы использовала флаг WRDE_ NOCMD, чтобы запретить замену команд.

WRDE_NOSPACE

Было невозможно зарезервировать память для результат. В этом случае, wordexp может сохранять часть результатов, столько, сколько она смогла зарезервировать памяти.

WRDE_SYNTAX

Имелась синтаксическая ошибка во входной строке. Например, несогласованные кавычки – синтаксическая ошибка.

`void wordfree (wordexp_t *word-vector-ptr)` (функция)

Освободит память, используемую для строки слов и вектора, на который указывает * word-vector-ptr. Она не освобождает структуру *word-vector-ptr непосредственно, а только другие данные, на которые она указывает.

Флаги для Разложения Слова

Этот раздел описывает флаги, которые Вы можете определять в аргументе flags wordexp. Выберите флаги, которые Вы хотите, и объединяете их оператором |.

WRDE_APPEND

Добавляет слова этого разложения к вектору слов, произведенных предыдущими обращениями к wordexp.

Для этого Вы не должны изменять содержимое структуры вектора слов между обращениями к wordexp. И, если Вы устанавливаете WRDE_DOOFFS в первом обращении к wordexp, Вы должны также установить его, когда Вы добавляете.

WRDE_DOOFFS

Оставляет пустое место в начале вектора слов. We_offs поле говорит, сколько места оставить. Пустое место содержит пустые указатели.

WRDE_NOCMD

Не делает замену команд; при попытке замены команды, сообщаает об ошибке.

WRDE_REUSE

Многократно использует вектор слов, сделанный предыдущим обращением к wordexp. Вместо того, чтобы зарезервировать новый вектор слов, это обращение к wordexp использует вектор, который уже существует (увеличивая его в случае необходимости).

Обратите внимание, что вектор может перемещаться в памяти, так что небезопасно хранить старый указатель и использовать его снова после вызова wordexp. Вы должны сохранять we_pathv после каждого обращения.

WRDE_SHOWERR

Покажет любые сообщения об ошибках.

WRDE_UNDEF

Если ввод относится к переменной оболочки которая не определена, выдает ошибку.

Пример Wordexp

Вот пример использования wordexp, чтобы рзложить отдельные строки и использования результатов чтобы выполнить команду оболочки. Он также показывает использование WRDE_APPEND, чтобы добавлять разложения и wordfree, чтобы освободить место, размещенное wordexp.

```
int
expand_and_execute(const char*program,const char*options)
{
 wordexp_t result;
 pid_t pid
 int status, i;
 switch (wordexp (program, &result, 0))
 {
         case 0:
                 break;
         case WRDE_NOSPACE:
                 wordfree (&result);
         default:
                 return -1;
 }
 for (i = 0; args[i]; i++)
 {
         if (wordexp (options, &result, WRDE_APPEND))
         {
                 wordfree (&result);
                 return -1;
         }
 }
 pid = fork ();
 if (pid == 0)
 {
         execv (result.we_wordv[0],
   result.we_wordv);
```

```
                exit (EXIT_FAILURE);
        }
        else if (pid < 0)
                status = -1;
        else
                 if (waitpid (pid, &status, 0) != pid)
        status = -1;
        wordfree (&result);
        return status;
```

Практически, т. к. wordexp работает, выполняя подоболочку, было бы быстрее сделать это, связывая строки с пробелами между ними и выполняя это как команду оболочки, используюя " sh -c ".

Вперед Назад Содержание