Как читать из файла или стандартного ввода в Bash

Спрошено 12 лет, 9 месяцев назад Изменено 7 месяцев назад Просмотрено 529 тыс. раз

333

Следующий Perl-скрипт (`my.pl`) может читать либо из файла в аргументах командной строки, либо из стандартного ввода (STDIN):

```
while (<>) {
   print($_);
}
```

`perl my.pl` будет считываться из стандартного ввода, в то время как `perl my.pl a.txt` будет считываться из `a.txt` . Это очень удобно.

Есть ли эквивалент в Bash?

bash stdin

Поделиться
Улучшить этот вопрос
Подписаться

отредактировано 21 июня 2021 г. в 12:45
Peter Mortensen
31.2k 22 109 132

спрошено 8 августа 2011 в 9:27
Даганг Вэй
25.8k 26 92 139

22 Ответа

Отсортировано по: Highest score (default)

Не нашли ответ? Задайте вопрос на Stack Overflow на русском.

548

Следующее решение выполняет чтение из файла, если скрипт вызывается с именем файла в качестве первого параметра `$1` и в противном случае из стандартного ввода.

```
while read line
do
  echo "$line"
done < "${1:-/dev/stdin}"
```

Подстановка `${1:-...}` выполняется `$1` , если она определена. В противном случае используется имя файла стандартного ввода собственного процесса.

Поделиться
Улучшить этот ответ
Подписаться

отредактировано 21 июня 2021 г. в 12:48
Peter Mortensen
31.2k 22 109 132

ответил 12 августа 2011 г. в 19:44
Fritz G. Mehner
17k 3 35 42

1 Отлично, это работает. Другой вопрос, почему вы добавляете для этого кавычки? "${1: - / proc / $ {$} / fd / 0}" – Даганг Вэй 13 августа 2011 в 8:02

19 Имя файла, которое вы указываете в командной строке, может содержать пробелы. –Fritz G. Mehner 13 августа 2011 в 9:12

8 Есть ли какая-либо разница между использованием `/proc/$$/fd/0` и `/dev/stdin` ? Я заметил, что последнее кажется более распространенным и выглядит более простым. –ноу-хау 14 января 2015 в 23:24

28 Лучше добавить `-r` в свою `read` команду, чтобы она случайно не поглощала `\` символы; используйте `while IFS= read -r line` для сохранения начальных и завершающих пробелов. –mklement0 28 февраля 2015 в 23:34

5 В подавляющем большинстве случаев вам следует избегать этого. Если все, что вы хотите сделать, это повторить ввод обратно в вывод, `cat` уже делает это. Очень часто обработка может выполняться в Awk-скрипте, а цикл shell `while read` только усложняет дело. Очевидно, что бывают ситуации, когда вам действительно нужно обрабатывать строку за раз из файла в цикле командной строки, но если вы только что нашли этот ответ в Google, вы должны знать, что это обычный антипаттерн для новичков. –tripleee 27 июля 2016 в 8:54

157

Возможно, самым простым решением является перенаправление стандартного ввода с помощью оператора перенаправления слияния:

```
#!/bin/bash
less <&0
```

Стандартным вводом является нулевой файловый дескриптор. Приведенный выше ввод, передаваемый вашему bash-скрипту, преобразуется в стандартный ввод less.

Узнайте больше о перенаправлении файлового дескриптора.

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 21 июня 2021 г. в 12:55

Peter Mortensen
31.2k 22 109 132

ответил 10 мая 2013 г. в 20:48

Райан Баллантайн
4,084 3 26 27

22 В этой ситуации использование `<&0` бесполезно - ваш пример будет работать одинаково с ним или без него - по-видимому, инструменты, которые вы вызываете из скрипта bash, по умолчанию видят тот же stdin, что и сам скрипт (если только скрипт не использует его первым). –mklement0 28 февраля 2015 в 23:43

@mkelement0 Итак, если инструмент считывает половину входного буфера, получит ли следующий инструмент, который я вызову, остальное? –Асад Саидуддин 3 мая 2017 в 16:28

1 "Отсутствует имя файла ("меньше --справка" для справки)", когда я это делаю... Ubuntu 16.04 –Омаротман 31 июля 2017 в 17:11

6 где часть "или из файла" в этом ответе? –Себастьян 2 августа 2017 в 19:13

Именно то, что мне было нужно! Я также отправил любые аргументы CLI: `less "$@" <&0`
–KingBob 7 февраля 2018 в 0:22

145

Вот самый простой способ:

```
#!/bin/sh
cat -
```

Использование:

```
$ echo test | sh my_script.sh
test
```

Чтобы присвоить *stdin* переменной, вы можете использовать: `STDIN=$(cat -)` или просто `STDIN=$(cat)` оператор as не нужен (согласно комментарию @mklement0).

Чтобы разобрать каждую строку из *стандартного ввода*, попробуйте следующий скрипт:

```
#!/bin/bash
while IFS= read -r line; do
  printf '%s\n' "$line"
done
```

Чтобы читать из файла или *stdin* (если аргумент отсутствует), вы можете расширить его до:

```
#!/bin/bash
file=${1--} # POSIX-compliant; ${1:--} can be used either.
while IFS= read -r line; do
  printf '%s\n' "$line" # Or: env POSIXLY_CORRECT=1 echo "$line"
done < <(cat -- "$file")
```

Примечания:

- `read -r` - Не используйте символ обратной косой черты каким-либо особым образом. Считайте, что каждая обратная косая черта является частью строки ввода.

- Без настройки `IFS` по умолчанию последовательности Space и Tab в начале и конце строк игнорируются (обрезаются).

- Используйте `printf` вместо `echo`, чтобы не печатать пустые строки, когда строка состоит из одного `-e`, `-n` или `-E`. Однако есть обходной путь с помощью `env POSIXLY_CORRECT=1 echo "$line"` который выполняет ваш *внешний* GNU `echo`, который его поддерживает. Смотрите: Как мне отобразить "-e"?

Смотрите: [Как читать stdin, когда аргументы не передаются?] в stackoverflow SE

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 23 мая 2017 г. в 12:02

Сообщество Бот
1 1

ответил 28 февраля 2015 г. в 19:58

kenorb
161k 93 694 760

1 Вы могли бы упростить `[ "$1" ] && FILE=$1 || FILE="-"` до `FILE=${1:--}`. (Замечание: лучше избегать использования всех прописных переменных *оболочки*, чтобы избежать конфликтов имен с переменными *окружения*.) –mklement0 28 февраля 2015 в 23:14

С удовольствием; на самом деле, `${1:--}` *является* совместимым с POSIX, поэтому он должен работать во всех POSIX-подобных оболочках. Что не будет работать во всех таких оболочках, так это подстановка процессов (`<(...)`); это будет работать в bash, ksh, zsh, но не в dash, например. Кроме того, лучше добавить `-r` в вашу `read` команду, чтобы она случайно не использовала `\` символы; добавляйте `IFS=` , чтобы сохранить начальные и конечные пробелы. –mklement0 28 февраля 2015 в 23:40

5 На самом деле ваш код по-прежнему ломается из-за `echo` : если строка состоит из `-e` , `-n` или `-E` , она не будет показана. Чтобы исправить это, вы должны использовать `printf` : `printf '%s\n' "$line"` . Я не включил это в свою предыдущую правку... слишком часто мои правки откатываются, когда я исправляю эту ошибку `:(` . –gniourf_gniourf 1 марта 2015 в 0:02

1 Нет, это не сбой. И `--` бесполезно, если первый аргумент равен `'%s\n'` –gniourf_gniourf 1 марта 2015 в 0:06

2 Ваш ответ меня устраивает (я имею в виду, что больше нет известных мне ошибок или нежелательных функций), хотя он не обрабатывает несколько аргументов, как это делает Perl. На самом деле, если вы хотите обрабатывать несколько аргументов, вы в конечном итоге напишете отличный ответ Джонатана Леффлера — на самом деле ваш был бы лучше, поскольку вы использовали бы `IFS=` with `read` и `printf` вместо `echo` . `:)` . –gniourf_gniourf 1 марта 2015 в 0:17

24

Я думаю, что это самый простой способ:

```
$ cat reader.sh
#!/bin/bash
while read line; do
  echo "reading: ${line}"
done < /dev/stdin
```

--

```
$ cat writer.sh
#!/bin/bash
for i in {0..5}; do
  echo "line ${i}"
done
```

--

```
$ ./writer.sh | ./reader.sh
reading: line 0
reading: line 1
reading: line 2
reading: line 3
reading: line 4
reading: line 5
```

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 21 июня 2021 г. в 12:56

Peter Mortensen
31.2k 22 109 132

ответил 26 марта 2014 г. в 19:25

Amir Mehler
4,380 3 30 38

5 Это не соответствует требованию poster для чтения либо из stdin, либо из аргумента file, это просто считывает из stdin. –nash 7 августа 2014 в 20:53

4 Оставляя в стороне обоснованное возражение @nash: `read` читает из stdin *по умолчанию*, так что в *этом нет* необходимости `< /dev/stdin` .–mklement0 28 февраля 2015 в 21:44

18

`echo` Решение добавляет новые строки всякий раз, когда `IFS` прерывается поток ввода. Ответ @fgm может быть немного изменен:

```
cat "${1:-/dev/stdin}" > "${2:-/dev/stdout}"
```

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 23 мая 2017 г. в 12:02

Сообщество Бот
1 1

ответил 19 августа 2011 г. в 20:19

Дэвид Саутер
8,134 2 38 53

Не могли бы вы, пожалуйста, объяснить, что вы подразумеваете под "echo solution добавляет новые строки всякий раз, когда IFS прерывает входной поток"? На случай, если вы имели в виду поведение `read` : while `read` *делает* потенциально разделенный на несколько токенов символами. содержащийся в `$IFS` , он возвращает только *одиночный* токен, если вы укажете только *единственное* имя переменной (но по умолчанию отделяет начальный и конечный пробелы). –mklement0 28 февраля 2015 в 23:19

@mklement0 я на 100% согласен с вами в поведении `read` and `$IFS` - `echo` сам добавляет новые строки без `-n` флага. "Утилита echo записывает любые указанные операнды, разделенные одним пустым символом (` ') и за которыми следует символ перевода строки (`\n'), в стандартный вывод". –Дэвид Саутер 1 марта 2015 в 16:36

Понял. Однако для эмуляции цикла Perl вам *нужен* завершающий элемент, `\n` добавленный с помощью `echo` : Perl `$_` *включает* окончание строки `\n` из прочитанной строки, в то время как bash `read` этого не делает. (Однако, как указывает @gniourf_gniourf в другом месте, более надежный подход заключается в использовании `printf '%s\n'` вместо `echo`). –mklement0 1 марта 2015 в 17:01

1 Это фантастическое решение, которое дает вашему скрипту возможность использоваться как `script.sh example` ИЛИ `cat example | script.sh` –shmup 18 августа 2022 в 5:12

13

Цикл Perl в вопросе считывает из *всех* аргументов имени файла в командной строке или из стандартного ввода, если файлы не указаны. Все ответы, которые я вижу, похоже, обрабатывают один файл или стандартный ввод, если файл не указан.

Хотя часто точно высмеивают как UUOC (бесполезное использование `cat`), бывают случаи, когда `cat` это лучший инструмент для работы, и можно утверждать, что это один из них:

```
cat "$@" |
while read -r line
do
    echo "$line"
done
```

Единственным недостатком этого является то, что он создает конвейер, работающий во вложенной оболочке, поэтому такие вещи, как присвоение переменных в `while` цикле, недоступны вне конвейера. `bash` Обходной путь заключается в замене процесса:

```
while read -r line
do
    echo "$line"
done < <(cat "$@")
```

Это оставляет `while` цикл запущенным в основной оболочке, поэтому переменные, установленные в цикле, доступны вне цикла.

Поделиться Улучшить этот ответ Подписаться

ответил 27 октября 2013 г. в 16:18

Джонатан Леффлер
743 тыс. 143 927 1,3 тыс.

1 Отличная мысль о *нескольких* файлах. Я не знаю, каковы будут последствия для ресурсов и производительности, но если вы не используете bash, ksh или zsh и поэтому не можете использовать замену процессов, вы могли бы попробовать here-doc с заменой команд (разбит на 3 строки) `>>EOF\n$(cat "$@")\nEOF` . Наконец, замечание: `while IFS= read -r line` это лучшее приближение к тому, что `while (<>)` делается в Perl (сохраняются начальные и конечные пробелы, хотя Perl также сохраняет конечные `\n`). –mklement0 28 февраля 2015 в 23:21

8

Поведение Perl, когда код, приведенный в OP, может не принимать ни одного или нескольких аргументов, и если аргумент состоит из одного дефиса `-` это понимается как stdin . Более того, всегда возможно указать имя файла с `$ARGV` . Ни один из приведенных до сих пор ответов на самом деле не имитирует поведение Perl в этих отношениях. Вот чистая возможность Bash. Хитрость заключается в том, чтобы использовать `exec` соответствующим образом.

```
#!/bin/bash

((${#})) || set -- -
while ((${#})); do
   { [[ $1 = - ]] || exec < "$1"; } &&
   while read -r; do
      printf '%s\n' "$REPLY"
   done
   shift
done
```

Имя файла доступно в `$1`.

Если аргументы не указаны, мы искусственно устанавливаем `-` в качестве первого позиционного параметра. Затем мы выполняем цикл с параметрами. Если параметр не задан `-`, мы перенаправляем стандартный ввод из filename с помощью `exec`. Если это перенаправление проходит успешно, мы выполняем цикл с помощью `while` цикла. Я использую стандартную `REPLY` переменную, и в этом случае вам не нужно выполнять сброс `IFS`. Если вам нужно другое имя, вы должны выполнить сброс `IFS` вот так (если, конечно, вы этого не хотите и знаете, что делаете):

```
while IFS= read -r line; do
    printf '%s\n' "$line"
done
```

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 28 февраля 2015 г. в 23:19

ответил 28 февраля 2015 г. в 23:09

gniourf_gniourf

45.8k 9 99 109

Это правильный ответ - недавно я узнал, как использовать exec для перенаправления stdout в указанный файл, я должен был знать, что его можно использовать для маршрутизации файла в stdin. Спасибо, что поделились своим ответом, извините, что он не получил той любви, которой заслуживает! –Дэвид Фаррелл 14 июля 2021 в 22:46

Более точно...

```
while IFS= read -r line ; do
    printf "%s\n" "$line"
done < file
```

3

Поделиться Улучшить этот ответ Подписаться

ответил 8 августа 2011 г. в 12:51

sorpigal

25.8k 8 58 77

5 Я предполагаю, что это по сути комментарий к stackoverflow.com/a/6980232/45375, а не ответ. Чтобы сделать комментарий явным: добавление `IFS=` и `-r` к `read` команде гарантирует,

что каждая строка будет прочитана *без изменений* (включая начальные и завершающие пробелы). –mklement0 28 февраля 2015 в 22:54

Пожалуйста, попробуйте следующий код:

3

```
while IFS= read -r line; do
    echo "$line"
done < file
```

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 28 февраля 2015 г. в 23:42

gniourf_gniourf

45.8k 9 99 109

ответил 8 августа 2011 г. в 9:40

Webthusiast

996 11 16

2 Обратите внимание, что даже с внесенными поправками это не будет считываться из стандартного ввода или из нескольких файлов, поэтому это не полный ответ на вопрос. (Также удивительно видеть две правки за считанные минуты, более чем через 3 года после первой отправки ответа.) –Джонатан Леффлер 1 марта 2015 в 0:07

@JonathanLeffler извините за редактирование такого старого (и не очень хорошего) ответа ... но я не мог вынести вида этого плохого `read` без `IFS=` и `-r`, и плохого `$line` без здоровых кавычек. –gniourf_gniourf 1 марта 2015 в 0:12

1 @gniourf_gniourf: Мне не нравятся `read -r` обозначения. IMO, POSIX ошибся; опция должна включать специальное значение для завершающих обратных косых черт, а не отключать его — чтобы существующие скрипты (до существования POSIX) не ломались, потому что `-r` был опущен. Я заметил, однако, что это было частью стандарта IEEE 1003.2 1992, который был самой ранней версией стандарта оболочки и утилит POSIX, но уже тогда он был отмечен как дополнение, так что это жалоба на давно упущенные возможности. Я никогда не сталкивался с проблемами, потому что мой код не использует `-r`; Должно быть, мне повезло. Игнорируйте меня на этот счет. –Джонатан Леффлер 1 марта 2015 в 0:27

1 @JonathanLeffler Я действительно согласен, что `-r` должно быть стандартным. Я согласен, что это маловероятно в случаях, когда его неиспользование приводит к проблемам. Хотя, неработающий код есть неработающий код. Сначала моя правка была вызвана этой плохой `$line` переменной, в которой сильно не хватало кавычек. Я исправил `read`, пока работал над этим. Я не исправлял `echo` потому что это тот вид редактирования, который откатывается. `:( .` –gniourf_gniourf 1 марта 2015 в 0:34

Как это работает? Что это за `IFS=` штука? Зачем это нужно? Есть некоторая информация в комментарии. –Peter Mortensen 21 июня 2021 в 12:51

Два основных способа:

3

- Либо передайте файлы аргументов и stdin в единый поток и обработайте это как stdin (*потоковый подход*)
- Или перенаправить stdin (и файлы аргументов) в именованный канал и обработать его как файл (*файловый подход*)

Потоковый подход

Незначительные изменения в предыдущих ответах:

- Используйте `cat`, а не `less`. Это быстрее, и вам не нужна разбивка на страницы.
- Используйте `$1` для чтения из файла с первым аргументом (если есть) или `$*` для чтения из всех файлов (если есть). Если эти переменные пусты, считайте из stdin (как это делает `cat`)

```
#!/bin/bash
cat $* | ...
```

Файловый подход

Запись в именованный канал немного сложнее, но это позволяет обрабатывать stdin (или файлы) как единый файл:

- Создайте канал с помощью `mkfifo`.
- Распараллелить процесс записи. Если именованный канал не считывается, в противном случае он может быть заблокирован.
- Для перенаправления stdin в подпроцесс (при необходимости в данном случае) используйте `<&0` (в отличие от того, что комментировали другие, здесь это *необязательно*).

```
#!/bin/bash
mkfifo /tmp/myStream
cat $* <&0 > /tmp/myStream &           # separate subprocess (!)
AddYourCommandHere /tmp/myStream       # process input like a file,
rm /tmp/myStream                       # cleaning up
```

Файловый подход: вариации

Создавайте именованный канал, только если не указаны аргументы. Это может быть более стабильным для чтения из файлов, поскольку именованные каналы иногда могут блокироваться.

```
#!/bin/bash
FILES=$*
if echo $FILES | egrep -v . >&/dev/null; then # if $FILES is empty
   mkfifo /tmp/myStream
   cat <&0 > /tmp/myStream &
   FILES=/tmp/myStream
fi
AddYourCommandHere $FILES     # do something ;)
if [ -e /tmp/myStream ]; then
   rm /tmp/myStream
fi
```

Кроме того, это позволяет вам выполнять итерации по файлам и stdin, а не объединять все в один поток:

```
for file in $FILES; do
    AddYourCommandHere $file
done
```

Поделиться Улучшить этот ответ Подписаться

ответил 8 апреля 2021 г. в 9:35

Chiarcos
354 2 11

Хороший ответ! Очень гибкий. Я хотел что-то для сортировки выходных данных `find` команды по дате файла, и это сработало отлично. –Марк Стюарт 24 октября 2023 в 23:58

3

```
#!/usr/bin/bash

if [ -p /dev/stdin ]; then
       #for FILE in "$@" /dev/stdin
    for FILE in /dev/stdin
    do
        while IFS= read -r LINE
        do
            echo "$@" "$LINE"   #print line argument and stdin
        done < "$FILE"
    done
else
    printf "[ -p /dev/stdin ] is false\n"
     #dosomething
fi
```

Выполняется:

```
echo var var2 | bash std.sh
```

Результат:

```
var var2
```

Выполняется:

```
bash std.sh < <(cat /etc/passwd)
```

Результат:

```
root:x:0:0::/root:/usr/bin/bash
bin:x:1:1::/:/usr/bin/nologin
daemon:x:2:2::/:/usr/bin/nologin
mail:x:8:12::/var/spool/mail:/usr/bin/nologin
```

Поделиться

ответил 8 декабря 2020 г. в 2:58

Улучшить этот ответ

Подписаться

отредактировано 9 июля 2021 г. в 16:28

Peter Mortensen
31.2k 22 109 132

vcatafesta
73 5

2

Следующее работает со стандартом `sh` (протестировано с Dash в Debian) и вполне читаемо, но это дело вкуса:

```
if [ -n "$1" ]; then
    cat "$1"
else
    cat
fi | commands_and_transformations
```

Подробнее: Если первый параметр непустой, то `cat` этот файл, иначе `cat` стандартный ввод. Затем вывод всего `if` оператора обрабатывается `commands_and_transformations`.

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 9 июля 2021 г. в 16:20

Peter Mortensen
31.2k 22 109 132

ответил 30 марта 2015 г. в 14:47

Список заметок
16.4k 11 57 101

1 ИМХО, лучший ответ so, потому что он указывает на истинное решение: `cat "${1:--}" | any_command`. Чтение переменных оболочки и их повторение может работать для небольших файлов, но масштабируется не так хорошо. –Андреас Шпиндлер 7 февраля 2017 в 17:49

1 `[ -n "$1" ]` Можно упростить до `[ "$1" ]`. –agc 1 мая 2017 в 2:06

2

Я объединил все приведенные выше ответы и создал функцию оболочки, которая соответствовала бы моим потребностям. Это из терминала Cygwin моих двух компьютеров с Windows 10, где у меня была общая папка между ними. Мне нужно уметь обрабатывать следующее:

- `cat file.cpp | tx`
- `tx < file.cpp`
- `tx file.cpp`

Если указано конкретное имя файла, мне нужно использовать то же имя файла при копировании. Если поток входных данных был передан по конвейеру, то мне нужно сгенерировать временное имя файла, содержащее час, минуту и секунды. Общая основная папка содержит вложенные папки с указанием дней недели. Это для организационных целей.

Вот, идеальный скрипт для моих нужд:

```
tx ()
{
  if [ $# -eq 0 ]; then
    local TMP=/tmp/tx.$(date +'%H%M%S')
    while IFS= read -r line; do
        echo "$line"
    done < /dev/stdin > $TMP
    cp $TMP //$OTHER/stargate/$(date +'%a')/
    rm -f $TMP
  else
    [ -r $1 ] && cp $1 //$OTHER/stargate/$(date +'%a')/ || echo "cannot read file"
  fi
}
```

Если есть какой-либо способ, который вы можете увидеть для дальнейшей оптимизации этого, я хотел бы знать.

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 9 июля 2021 г. в 16:26

Peter Mortensen
31.2k 22 109 132

ответил 28 сентября 2018 г. в 18:17

daparic
4,094 2 40 41

1

Код `${1:-/dev/stdin}` поймет только первый аргумент, поэтому вы можете использовать это:

```
ARGS='$*'
if [ -z "$*" ]; then
  ARGS='-'
fi
eval "cat -- $ARGS" | while read line
do
   echo "$line"
done
```

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 9 июля 2021 г. в 16: 21

Peter Mortensen
31.2k 22 109 132

ответил 8 октября 2015 г. в 15:10

Такахиро Онодера
11 1

1

Чтение из stdin в переменную или из файла в переменную.

В большинстве примеров в существующих ответах используются циклы, которые **немедленно повторяют каждую строку при чтении из stdin**. Это **может быть не то, что вы действительно хотите сделать.**

Во многих случаях вам нужно написать скрипт, вызывающий команду, которая принимает только аргумент file. Но в вашем скрипте вы можете захотеть также поддерживать stdin. В этом случае вам нужно сначала прочитать полный stdin, а затем предоставить его в виде файла.

Давайте посмотрим пример. Приведенный ниже скрипт печатает сведения о сертификате сертификата (в формате PEM), который передается либо как файл, либо через stdin.

```
# print-cert script

content=""
while read line
do
  content="$content$line\n"
done < "${1:-/dev/stdin}"
# Remove the last newline appended in the above loop
content=${content%\\n}

# Keytool accepts certificate only via a file, but in our script we fix this.
keytool -printcert -v -file <(echo -e $content)



# Read from file

cert-print mycert.crt

# Owner: CN=....
# Issuer: ....
# ....


# Or read from stdin (by pasting)

cert-print
#..paste the cert here and press enter
# Ctl-D

# Owner: CN=....
# Issuer: ....
# ....


# Or read from stdin by piping to another command (which just prints the cert(s)
). In this case we use openssl to fetch directly from a site and then print its
info.


echo "" | openssl s_client -connect www.google.com:443 -prexit 2>/dev/null \
| sed -n -e '/BEGIN\ CERTIFICATE/,/END\ CERTIFICATE/ p' \
| cert-print

# Owner: CN=....
# Issuer: ....
# ....
```

Поделиться Улучшить этот ответ Подписаться

ответил 28 марта 2022 г. в 13:43

Маринос
10.2k 6 68 106

Этот файл прост в использовании на терминале:

```
$ echo '1\n2\n3\n' | while read -r; do echo $REPLY; done
1
2
3
```

0

Поделиться Улучшить этот ответ Подписаться

ответил 5 июня 2020 г. в 1:55

cmcginty

115k 42 165 163

0

В качестве обходного пути вы можете использовать `stdin` устройство в каталоге */dev*:

```
....| для элемента в `cat / dev / stdin` ; выполнить echo $item ; готово
```

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 9 июля 2021 г. в 16:27

Peter Mortensen

31.2k 22 109 132

ответил 4 октября 2020 г. в 7:00

Алаа Ахмад М. Закария

253 4 11

0

С помощью...

```
while read line
do
    echo "$line"
done < "${1:-/dev/stdin}"
```

Я получил следующий вывод:

> Игнорируются 1265 символов стандартного ввода. Используйте "-stdin" или "-", чтобы указать, как обрабатывать конвейерный ввод.

Затем решил с помощью for:

```
Lnl=$(cat file.txt | wc -l)
echo "Last line: $Lnl"
nl=1

for num in `seq $nl +1 $Lnl`;
do
    echo "Number line: $nl"
    line=$(cat file.txt | head -n $nl | tail -n 1)
    echo "Read line: $line"
    nl=$[$nl+1]
done
```

Поделиться

Улучшить этот ответ

отредактировано 9 июля 2021 г. в 16:32

ответил 30 июня 2021 г. в 23:22

Подписаться

Peter Mortensen
31.2k 22 109 132

Ueliton Alves Dos Santos
1 2

0

Просто проверьте количество аргументов в вашем скрипте и проверьте, является ли первый аргумент (`$1`) файлом. Если false, используйте stdin `-` :

```
#!/bin/bash
[ $# -ge 1 -a -f "$1" ] && input="$1" || input="-"
cat $input
```

Смотрите этот вопрос.

Поделиться Улучшить этот ответ Подписаться

ответил 24 мая 2023 г. в 1:34

Юджин Ху
1,189 8 19

0

Ответ @gniourf_gniourf является наиболее правильным, но использует много bashisms и пропускает угловой регистр. Поскольку этот вопрос является лучшим результатом Google, вот версия, совместимая с POSIX:

```
#!/bin/sh

exec 3<&0

if [ $# -eq 0 ]; then
    set -- -
fi

for f in "$@"; do
    if { [ "$f" = - ] && exec <&3; } || exec < "$f"; then
        while IFS= read -r line; do
            printf '%s\n' "$line"
        done
    fi
done
```

или если вы хотите быть кратким:

```
#!/bin/sh
exec 3<&0
[ $# -eq 0 ] && set -- -
for f; do
    { { [ "$f" = - ] && exec <&3; } || exec < "$f"; } &&
    while IFS= read -r line; do
        printf '%s\n' "$line"
    done
done
```

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 13 сентября 2023 г. в 14:41

ответил 19 июля 2023 г. в 16:12

JM0

388 1 14

-1

Использование:

```
for line in `cat`; do
    something($line);
done
```

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 21 июня 2021 г. в 12:57

Peter Mortensen

31.2k 22 109 132

ответил 5 сентября 2014 г. в 21:03

Чарльз Купер

1

Вывод `cat` будет помещен в командную строку. Командная строка имеет максимальный размер. Также при этом будет считываться не построчно, а слово за словом. –Notinlist 1 мая 2017 в 11:16

-1

Я не нахожу ни один из этих ответов приемлемым. В частности, принятый ответ обрабатывает только первый параметр командной строки и игнорирует остальные. Программа Perl, которую он пытается эмулировать, обрабатывает все параметры командной строки. Таким образом, принятый ответ даже не отвечает на вопрос.

Другие ответы используют расширения Bash, добавляют ненужные команды 'cat', работают только для простого случая эхо-преобразования ввода в вывод или просто излишне сложны.

Однако я должен отдать им должное, потому что они подарили мне несколько идей. Вот полный ответ:

```
#!/bin/sh

if [ $# = 0 ]
then
        DEFAULT_INPUT_FILE=/dev/stdin
else
        DEFAULT_INPUT_FILE=
fi

# Iterates over all parameters or /dev/stdin
for FILE in "$@" $DEFAULT_INPUT_FILE
do
        while IFS= read -r LINE
        do
                # Do whatever you want with LINE here.
                echo $LINE
        done < "$FILE"
done
```

Поделиться

Улучшить этот ответ

Подписаться

отредактировано 9 июля 2021 г. в 16:24

Peter Mortensen
31.2k 22 109 132

ответил 5 марта 2018 г. в 17:11

Gungwald
142 1 8