

Преобразование абсолютного пути в относительный путь, заданный текущим каталогом, с помощью Bash

Спросил 14 лет, 6 месяцев назад Изменено 7 месяцев назад Просмотрено 208 тыс. раз



Пример:

378

```
absolute="/foo/bar"  
current="/foo/baz/foo"
```



```
# Magic
```



```
relative="../../../bar"
```



Как мне создать магию (надеюсь, не слишком сложный код...)?

Баш

оболочка

путь

относительный-путь

абсолютный-путь

Делиться Улучшить этот вопрос

Следовать

отредактировано 27 сен 2015 в 21:08



Питер Мортенсен

31,6 тыс. 22 109
132

задано 2 апреля 2010 г. в 1:42



Пол Тарьян

50,3 тыс. 59 175
214

8 Например (мой случай прямо сейчас) для указания относительного пути gcc, чтобы он мог генерировать относительную отладочную информацию, пригодную для использования даже при изменении исходного пути. – Я предлагаю 19 сен 2012 в 16:07

2 Похожий вопрос был задан на U&L: unix.stackexchange.com/questions/100918/... В одном из ответов (@Gilles) упоминается инструмент symlinks, который может облегчить работу с этой проблемой. – слм 13 ноя 2013 в 0:56 ✎

Я удивлен, что для этого нет инструмента (или опции) `realpath`. – Шридхар Сарнобат 15 июня 2023 г. в 20:09

Сортировать по:

25 ответов

Наивысший балл (по умолчанию)



Я думаю, проще всего использовать `realpath` GNU coreutils 8.23:

386

```
$ realpath -s --relative-to="$file1" "$file2"
```



Например:





```
$ realpath -s --relative-to=/usr/bin/nmap /tmp/testing
../../../../tmp/testing
```

Флаг `-s` гарантирует, что символические ссылки не будут расширяться.

Делиться Улучшить этот ответ

отредактировано 5 мар в 21:10

ответил 15 февр. 2015 г. в 4:59

Следовать



Флимм

147 тыс.

48

268

283



модуль0

4,168

1

13

11

8 Жаль, что пакет устарел в Ubuntu 14.04 и не имеет опции `--relative-to`.

– [округ Колумбия](#) 21 июл 2016 в 15:18

5 Отлично работает на Ubuntu 16.04 – [кайхорстманн](#) 13 февр. 2017 г. в 14:42

13 `$ realpath --relative-to="${PWD}" "$file"` полезно, если вам нужны пути относительно текущего рабочего каталога. – [dcoles](#) 22 авг 2017 в 19:11

7 Как подразумевал [@PatrickB.](#), `--relative-to=...` ожидает каталог и НЕ проверяет. Это означает, что вы получаете дополнительный `../`, если запрашиваете путь относительно файла (как, похоже, делает этот пример, потому что `/usr/bin` редко или никогда не содержит каталогов и `nmap` обычно является двоичным) – [IBBard](#) 13 марта 2018 г. в 20:07

2 Обратите внимание, что это также расширяет символические ссылки. Чтобы получить относительный путь самой символической ссылки, используйте `realpath -s --relative-to ...` – [Алекс](#) 29 сен 2023 г. в 8:10



195



дает:

```
../../../../bar
```



Делиться Улучшить этот ответ

отредактировано 25 июля 2014 г. в 19:12

ответил 5 сен 2011 в 7:50

Следовать



Сэм Р.

16.4k

14

72

124



xni

2,065

1

12

2

11 Это работает, и это заставляет альтернативы выглядеть смешными. Это бонус для меня xD – [хасвн](#) 19 марта 2012 г. в 16:54

38 +1. Ладно, ты сжульничал... но это слишком хорошо, чтобы этим не воспользоваться! `relpath(){ python -c "import os.path; print os.path.relpath('$1', '${2:-$PWD}')" ; }` – [МестреЛион](#) 4 апр. 2012 г. в 18:27 ✎

5 К сожалению, это доступно не везде: `os.path.relpath` – это нововведение в Python 2.6. – [Чэнь Леви](#) 15 ноября 2012 г., 11:01 ✎

16 [@ChenLevy](#): Python 2.6 был выпущен в 2008 году. Трудно поверить, что он не был общедоступным в 2012 году. – [МестреЛион](#) 23 авг. 2013 г., 7:29

16 python -c 'import os, sys; print(os.path.relpath(*sys.argv[1:]))' работает максимально естественно и надежно. – [не живи](#) 23 июл 2015 в 20:36 ✎



Он встроен в [Perl](#) с 2001 года, поэтому работает практически на любой системе, которую только можно себе представить, даже на [VMS](#) .

32



```
perl -le 'use File::Spec; print File::Spec->abs2rel(@ARGV)' FILE BASE
```



Кроме того, решение легко понять.



Итак, для вашего примера:

```
perl -le 'use File::Spec; print File::Spec->abs2rel(@ARGV)' $absolute $current
```

...было бы отлично.

Делиться Улучшить этот ответ

Следовать

отредактировано 25 окт. 2022 г. в 21:16

ответил 14 июня 2013 г. в 14:15



[Кен Уильямс](#)

23.8 тыс. 11 96 151



[Эрик Аронести](#)

12,6 тыс. 6 68 46

3 say не был доступен в Perl для использования в качестве журнала, но его можно эффективно использовать здесь. perl -MFile::Spec -E 'say File::Spec->abs2rel(@ARGV)' – [Уильям Перселл](#) 17 авг. 2014 г., 13:45 ✎

1 +1, но см. также [этот похожий ответ](#) , который старше (февраль 2012 г.). Прочтите также соответствующие комментарии от [Уильяма Перселла](#) . Моя версия – это две командные строки: perl -MFile::Spec -e 'print File::Spec->abs2rel(@ARGV)' "\$target" и perl -MFile::Spec -e 'print File::Spec->abs2rel(@ARGV)' "\$target" "\$origin" . Первый однострочный скрипт [perl](#) использует один аргумент (origin – текущий рабочий каталог). Второй однострочный скрипт [perl](#) использует два аргумента. – [охо](#) 1 окт. 2014 г. в 17:03 ✎

5 Это должен быть общепринятый ответ. его perl можно найти почти везде, хотя ответ по-прежнему однострочный. – [Дмитрий Гинзбург](#) 8 мая 2015 г., 9:52

@WilliamPursell Того же самого можно добиться с помощью -l переключателя командной строки, я добавил его в пример. – [Кен Уильямс](#) 25 окт. 2022 г. в 21:17



Это исправленное, полностью функциональное улучшение лучшего на данный момент решения от @rini (которое, к сожалению, обрабатывает только несколько случаев)

32



Напоминание: -z проверьте, имеет ли строка нулевую длину (=пуста), и проверьте, не -n является ли строка пустой.



```
# both $1 and $2 are absolute paths beginning with /
# returns relative path to $2/$target from $1/$source
```

```

source=$1
target=$2

common_part=$source # for now
result="" # for now

while [[ "${target#$common_part}" == "${target}" ]]; do
    # no match, means that candidate common part is not correct
    # go up one level (reduce common part)
    common_part="$(dirname $common_part)"
    # and record that we went back, with correct / handling
    if [[ -z $result ]]; then
        result=".."
    else
        result="../$result"
    fi
done

if [[ $common_part == "/" ]]; then
    # special case for root (no common path)
    result="$result/"
fi

# since we now have identified the common part,
# compute the non-common part
forward_part="${target#$common_part}"

# and now stick all parts together
if [[ -n $result ]] && [[ -n $forward_part ]]; then
    result="$result$forward_part"
elif [[ -n $forward_part ]]; then
    # extra slash removal
    result="${forward_part:1}"
fi

echo $result

```

Тестовые случаи:

```

compute_relative.sh "/A/B/C" "/A"          --> "../.."
compute_relative.sh "/A/B/C" "/A/B"        --> ".."
compute_relative.sh "/A/B/C" "/A/B/C"      --> ""
compute_relative.sh "/A/B/C" "/A/B/C/D"    --> "D"
compute_relative.sh "/A/B/C" "/A/B/C/D/E"  --> "D/E"
compute_relative.sh "/A/B/C" "/A/B/D"      --> "../D"
compute_relative.sh "/A/B/C" "/A/B/D/E"    --> "../D/E"
compute_relative.sh "/A/B/C" "/A/D"        --> "../../../D"
compute_relative.sh "/A/B/C" "/A/D/E"      --> "../../../D/E"
compute_relative.sh "/A/B/C" "/D/E/F"      --> "../../../D/E/F"

```

Делиться Улучшить этот ответ

Следовать

отредактировано 25 февр. 2023
г. в 8:01



Майкл Лихс

8,120 17 55 89

ответил 19 сен 2012 в 15:58



Я предлагаю

19,6 тыс. 12 80 98

1 Интегрирована в библиотеку оболочки offirmo github.com/Offirmo/offirmo-shell-lib, функция «OSL_FILE_find_relative_path» (файл «osl_lib_file.sh») – Я предлагаю 3 дек. 2012 г. в 14:53

2 +1. Его можно легко заставить обрабатывать любые пути (не только абсолютные пути, начинающиеся с /), заменив `source=$1; target=$2` на `source=$(realpath $1);`

target=\$(realpath \$2) - [Джош Келли](#) 11 сен 2013 в 12:54

- 2 @Джош, конечно, при условии, что `dirs` действительно существует... что было неудобно для модульных тестов ;) Но в реальном использовании да, `realpath` рекомендуется, или `source=$(readlink -f $1)` т. д., если `realpath` недоступен (не стандартный)
- [Я предлагаю](#) 11 сен 2013 в 13:06

Я определил `$source` и `$target` следующим образом: `` if [[-e $1]]; then source=$(readlink -f $1); else source=$1; fi if [[-e $2]]; then target=$(readlink -f $2); else target=$2; fi`` Таким образом, функция могла бы обрабатывать как реальные/существующие относительные пути, так и вымышленные каталоги.

- [Натан С. Уотсон-Хэйг](#) 2 июля 2014 г., 4:01

- 1 @NathanS.Watson-Haigh Еще лучше, я недавно обнаружил, `readlink` что есть `-m` опция, которая делает именно это ;) - [Я предлагаю](#) 2 июл 2014 в 12:36 ✎



29



```
#!/bin/bash
```

```
# both $1 and $2 are absolute paths
```

```
# returns $2 relative to $1
```

```
source=$1
```

```
target=$2
```

```
common_part=$source
```

```
back=
```

```
while [ "${target#$common_part}" = "${target}" ]; do
```

```
    common_part=$(dirname $common_part)
```

```
    back="$../${back}"
```

```
done
```

```
echo ${back}${target#$common_part}/
```

Делиться Улучшить этот ответ Следовать

ответил 17 февр. 2011 г. в 10:40



бобы

315 3 2

Замечательный скрипт – короткий и чистый. Я применил правку (ожидая рецензирования): `common_part=$source/ common_part=$(dirname $common_part)/ echo ${back}${target#$common_part}` Существующий скрипт не сработает из-за несоответствующего соответствия в начале имени каталога при сравнении, например: `"/foo/bar/baz"` с `"/foo/barsucks/bonk"`. Перемещение слеша в `var` и из финального `eval` исправляет эту ошибку. - [Йквенгер](#) 10 июня 2011 г. в 15:31 ✎

- 3 Этот скрипт просто не работает. Проваливает один простой тест "на один каталог вниз". Правки `jsweniger` работают немного лучше, но имеют тенденцию добавлять дополнительный `"/"`. - [ТАК воняло](#) 23 июл 2011 в 13:01

- 1 В некоторых случаях у меня это не получается, если в аргументе есть завершающий символ `"/"`; например, если `$1="$HOME/"` и `$2="$HOME/temp"`, возвращается `"/home/user/temp/"`, но если `$1=$HOME`, то он правильно возвращает относительный путь `"temp"`. Поэтому и `source=$1`, и `target=$2` можно "очистить" с помощью `sed` (или с помощью подстановки переменных `bash`, но это может быть излишне непрозрачно), например `=> source=$(echo "${1}" | sed 's/\/*$//')` - [Майкл](#) 28 мар. 2012 г. в 22:04

- 1 Небольшое улучшение: вместо установки источника/цели напрямую в `$1` и `$2`, сделайте: `source=$(cd $1; pwd) target=$(cd $2; pwd)`. Таким образом, он правильно обрабатывает

пути с . и .. – [Джозеф Гарвин](#) 4 апр. 2012 г. в 15:11

- 4 Несмотря на то, что это ответ с наибольшим количеством голосов, у этого ответа есть много ограничений, поэтому публикуется так много других ответов. Вместо этого посмотрите другие ответы, особенно тот, который показывает тестовые случаи. И, пожалуйста, поддержите этот комментарий! – [Я предлагаю](#) 1 окт. 2012 г. в 9:25



22



```
#!/bin/bash
s=$(cd ${1%}/;pwd); d=$(cd $2;pwd); b=; while [ "${d#s/}" == "${d}" ]
do s=$(dirname $s);b="../${b}"; done; echo ${b}${d#s/}
```



Я взял ответ от [pini](#) и еще несколько идей

Примечание : для этого требуется, чтобы оба пути были существующими папками. Файлы **не** будут работать.

Делиться Улучшить этот ответ

Следовать

отредактировано 9 дек. 2019 г. в 12:30



Teck-фрик
127 8

ответил 20 июля 2014 г. в 8:32



Алекс Рош
3,233 1 34 39

- 2 идеальный ответ: работает с /bin/sh, не требует readlink, python, perl -> отлично подходит для легких/встраиваемых систем или консоли Windows Bash – [Франсуа](#) 28 окт. 2014 г. в 13:57

- 2 К сожалению, для этого необходимо наличие пути, что не всегда желательно.
– [drwatsoncode](#) 10 окт. 2015 г. в 6:08

Божественный ответ. Я полагаю, что cd-pwd – это для разрешения ссылок? Отличная игра в гольф! – [Teck-фрик](#) 9 дек. 2019 г. в 8:10



17



Python `os.path.relpath` как функция оболочки

Цель этого `relpath` упражнения – имитировать `os.path.relpath` функцию Python 2.7 (доступную с версии Python 2.6, но работающую правильно только в 2.7), как предложено [xni](#) . Вследствие этого некоторые результаты могут отличаться от функций, представленных в других ответах.

(Я не тестировал с переводами строк в путях просто потому, что это нарушает проверку, основанную на вызове `python -c` из ZSH. Это, безусловно, было бы возможно, если бы пришлось приложить некоторые усилия.)

Что касается «магии» в Bash, я давно отказался от поиска магии в Bash, но с тех пор я нашел всю необходимую мне магию, и даже больше, в ZSH.

Поэтому я предлагаю два варианта реализации.

Первая реализация нацелена на полную *совместимость с POSIX*. Я тестировал ее `/bin/dash` на Debian 6.0.6 «Squeeze». Она также отлично работает на `/bin/sh` OS X 10.8.3, которая на самом деле является Bash версии 3.2, притворяющейся оболочкой POSIX.

Вторая реализация – это функция оболочки ZSH, которая устойчива к множественным слешам и другим помехам в путях. Если у вас есть ZSH, это рекомендуемая версия, даже если вы вызываете ее в форме скрипта, представленной ниже (т. е. с помощью shebang `#!/usr/bin/env zsh`) из другой оболочки.

Наконец, я написал скрипт ZSH, который проверяет вывод команды, `relpath` найденной в `$PATH` с учетом тестовых случаев, предоставленных в других ответах. Я добавил немного остроты этим тестам, добавив несколько пробелов, табуляций и знаков препинания, таких как `! ? *` здесь и там, а также добавил еще один тест с экзотическими символами UTF-8, найденными в [vim-powerline](#).

Функция оболочки *POSIX*

Во-первых, функция оболочки, совместимая с POSIX. Она работает с различными путями, но не очищает множественные слеша и не разрешает символические ссылки.

```
#!/bin/sh
relpath () {
    [ $# -ge 1 ] && [ $# -le 2 ] || return 1
    current="${2:+"$1"}"
    target="${2:-"$1"}"
    [ "$target" != . ] || target=/
    target="/${target##/}"
    [ "$current" != . ] || current=/
    current="${current:=/}"
    current="/${current##/}"
    appendix="${target##/}"
    relative=""
    while appendix="${target#"${current}"/}"
    [ "$current" != '/' ] && [ "$appendix" = "$target" ]; do
        if [ "$current" = "$appendix" ]; then
            relative="${relative:-.}"
            echo "${relative#}/"
            return 0
        fi
        current="${current%/*}"
        relative="$relative${relative:+/}.."
    done
    relative="$relative${relative:+${appendix:+/}}${appendix#}/"
    echo "$relative"
}
relpath "$@"
```

Функция оболочки ZSH

Теперь более надежная `zsh` версия. Если вы хотите, чтобы она разрешала аргументы в реальные пути а-ля `realpath -f` (доступно в пакете Linux `coreutils`),

замените `:a` в строках 3 и 4 на `:A`.

Чтобы использовать это в `zsh`, удалите первую и последнюю строку и поместите ее в каталог, который находится в вашей `$FPATH` переменной.

```
#!/usr/bin/env zsh
relpath () {
    [[ $# -ge 1 ]] && [[ $# -le 2 ]] || return 1
    local target=${${2:-$1}:a} # replace `:a` by `:A` to resolve symlinks
    local current=${${${2:+$1}:-$PWD}:a} # replace `:a` by `:A` to resolve
symlinks
    local appendix=${target#/}
    local relative=''
    while appendix=${target#$current/}
    do
        [[ $current != '/' ]] && [[ $appendix = $target ]]; do
            if [[ $current = $appendix ]]; then
                relative=${relative:-.}
                print ${relative#/}
                return 0
            fi
            current=${current%/*}
            relative="$relative${relative:+/}.."
        done
        relative+=${relative:+${appendix:+/}}${appendix#/}
        print $relative
    }
    relpath "$@"
}
```

Тестовый сценарий

Наконец, тестовый скрипт. Он принимает одну опцию, а именно `-v` включение подробного вывода.

```
#!/usr/bin/env zsh
set -eu
VERBOSE=false
script_name=$(basename $0)

usage () {
    print "\n    Usage: $script_name SRC_PATH DESTINATION_PATH\n" >&2
    exit ${1:=1}
}

vrb () { $VERBOSE && print -P ${(%):@} || return 0; }

relpath_check () {
    [[ $# -ge 1 ]] && [[ $# -le 2 ]] || return 1
    target=${${2:-$1}}
    prefix=${${${2:+$1}:-$PWD}}
    result=$(relpath $prefix $target)
    # Compare with python's os.path.relpath function
    py_result=$(python -c "import os.path; print os.path.relpath('$target',
'$prefix')")
    col='%F{green}'
    if [[ $result != $py_result ]] && col='%F{red}' || $VERBOSE; then
        print -P "${col}Source: '$prefix'\nDestination: '$target'%f"
        print -P "${col}relpath: ${(%):result}%f"
        print -P "${col}python:  ${(%):py_result}%f\n"
    fi
}
```



```

run_checks () {
    print "Running checks..."

    relpath_check '/' a b/â/≡*/!' '/' a b/â/≡/xää/?'

    relpath_check '/' '/A'
    relpath_check '/A' '/'
    relpath_check '/' & / !/*/\E' '/'
    relpath_check '/' '/' & / !/*/\E'
    relpath_check '/' & / !/*/\E' '/' & / !/?/\E/F'
    relpath_check '/X/Y' '/' & / !/C/\E/F'
    relpath_check '/' & / !/C' '/A'
    relpath_check '/A / !/C' '/A /B'
    relpath_check '/Â/ !/C' '/Â/ !/C'
    relpath_check '/' & /B / C' '/' & /B / C/D'
    relpath_check '/' & / !/C' '/' & / !/C/\E'
    relpath_check '/Â/ !/C' '/Â/ !/D'
    relpath_check '/.A /*B/C' '/.A /*B/\E'
    relpath_check '/' & / !/C' '/' & /D'
    relpath_check '/' & / !/C' '/' & /\E'
    relpath_check '/' & / !/C' '/' /\E/F'

    relpath_check /home/part1/part2 /home/part1/part3
    relpath_check /home/part1/part2 /home/part4/part5
    relpath_check /home/part1/part2 /work/part6/part7
    relpath_check /home/part1 /work/part1/part2/part3/part4
    relpath_check /home /work/part2/part3
    relpath_check / /work/part2/part3/part4
    relpath_check /home/part1/part2 /home/part1/part2/part3/part4
    relpath_check /home/part1/part2 /home/part1/part2/part3
    relpath_check /home/part1/part2 /home/part1/part2
    relpath_check /home/part1/part2 /home/part1
    relpath_check /home/part1/part2 /home
    relpath_check /home/part1/part2 /
    relpath_check /home/part1/part2 /work
    relpath_check /home/part1/part2 /work/part1
    relpath_check /home/part1/part2 /work/part1/part2
    relpath_check /home/part1/part2 /work/part1/part2/part3
    relpath_check /home/part1/part2 /work/part1/part2/part3/part4
    relpath_check home/part1/part2 home/part1/part3
    relpath_check home/part1/part2 home/part4/part5
    relpath_check home/part1/part2 work/part6/part7
    relpath_check home/part1 work/part1/part2/part3/part4
    relpath_check home work/part2/part3
    relpath_check . work/part2/part3
    relpath_check home/part1/part2 home/part1/part2/part3/part4
    relpath_check home/part1/part2 home/part1/part2/part3
    relpath_check home/part1/part2 home/part1/part2
    relpath_check home/part1/part2 home/part1
    relpath_check home/part1/part2 home
    relpath_check home/part1/part2 .
    relpath_check home/part1/part2 work
    relpath_check home/part1/part2 work/part1
    relpath_check home/part1/part2 work/part1/part2
    relpath_check home/part1/part2 work/part1/part2/part3
    relpath_check home/part1/part2 work/part1/part2/part3/part4

    print "Done with checks."
}
if [[ $# -gt 0 ]] && [[ $1 = "-v" ]]; then
    VERBOSE=true
    shift
fi
if [[ $# -eq 0 ]]; then
    run_checks
else

```

```
VERBOSE=true
relpath_check "$@"
fi
```

Делиться Улучшить этот ответ
Следовать

отредактировано 29 июля 2022 г.
в 16:55

ответил 16 февр. 2013 г. в
19:12



говорящий с тенью
13,7 тыс. 5 59 110



Саймон Хеймлихер
391 3 7

2 Боюсь, не работает, если первый путь заканчивается на / «I'm Voop». – [Нолдорин](#) 10
февр. 2015 г. в 3:31

@Noldorin версия Zsh, похоже, справляется с этим нормально. – [говорящий с тенью](#) 29
июл. 2022 в 16:32

@shadowtalker Я имел в виду версию POSIX (использующую Bash). – [Нолдорин](#) 29 июл. 2022
в 22:47



13



```
#!/bin/sh
```

```
# Return relative path from canonical absolute dir path $1 to canonical
# absolute dir path $2 ($1 and/or $2 may end with one or no "/").
# Does only need POSIX shell builtins (no external command)
relPath () {
    local common path up
    common=${1%/} path=${2%/}/
    while test "${path#"${common}"/}" = "$path"; do
        common=${common%/*} up=../$up
    done
    path=$up${path#"${common}"/}; path=${path%/}; printf %s "${path:-..}"
}
```

```
# Return relative path from dir $1 to dir $2 (Does not impose any
# restrictions on $1 and $2 but requires GNU Core Utility "readlink"
# HINT: busybox's "readlink" does not support option '-m', only '-f'
# which requires that all but the last path component must exist)
relpath () { relPath "$(readlink -m "$1")" "$(readlink -m "$2")"; }
```

Вышеприведенный скрипт оболочки был вдохновлен [pini](#) (Спасибо!). Он вызывает ошибку в модуле подсветки синтаксиса Stack Overflow (по крайней мере, в моем кадре предварительного просмотра). Поэтому, пожалуйста, проигнорируйте, если подсветка некорректна.

Некоторые примечания:

- Устранены ошибки и улучшен код без значительного увеличения длины и сложности кода.
- Добавьте функциональность в функции для удобства использования
- Функции сохранены совместимыми с POSIX, чтобы они (должны) работать со всеми оболочками POSIX (протестировано с dash, bash и zsh в Ubuntu Linux 12.04)

- Использовать только локальные переменные, чтобы избежать затирания глобальных переменных и загрязнения глобального пространства имен.
- Оба пути к каталогам НЕ должны существовать (требование для моего приложения)
- Имена путей могут содержать пробелы, специальные символы, управляющие символы, обратные косые черты, символы табуляции, ' , " , ? , * , [,] и т. д.
- Основная функция «relPath» использует только встроенные функции оболочки POSIX, но требует канонические абсолютные пути к каталогам в качестве параметров.
- Расширенная функция "relpath" может обрабатывать произвольные пути к каталогам (также относительные, неканонические), но требует внешней утилиты ядра GNU "readlink"
- Вместо встроенного «echo» мы использовали встроенный «printf» по двум причинам:
 - Из-за противоречивых исторических реализаций встроенной функции «echo» она ведет себя по-разному в разных оболочках -> [POSIX рекомендует отдавать предпочтение printf, а не echo](#) .
 - Встроенная функция «echo» некоторых оболочек POSIX будет [интерпретировать некоторые последовательности обратной косой черты](#) и, таким образом, повреждать имена путей, содержащие такие последовательности.
- Чтобы избежать ненужных преобразований, пути используются в том виде, в котором они возвращаются и ожидаются утилитами оболочки и ОС (например, cd, ln, ls, find, mkdir; в отличие от "os.path.relpath" в Python, который будет интерпретировать некоторые последовательности обратной косой черты).
- За исключением упомянутых последовательностей обратного следа последняя строка функции «relPath» выводит имена путей, совместимые с Python:

```
path=$up${path#"$common"/}; path=${path%/*}; printf %s "${path:-.}"
```

Последнюю строку можно заменить (и упростить) строкой

```
printf %s "$up${path#"$common"/}"
```

Я предпочитаю последнее, потому что

1. Имена файлов можно напрямую добавлять к путям каталогов, полученным с помощью relPath, например:

```
ln -s "$(relpath "<fromDir>" "<toDir>")<file>" "<fromDir>"
```

2. Символические ссылки в том же каталоге, созданные с помощью этого метода, не имеют уродливого "./" префикса к имени файла.

- Если вы нашли ошибку, пожалуйста, свяжитесь с linuxball (at) gmail.com, и я постараюсь ее исправить.
- Добавлен набор регрессионных тестов (также совместимый с оболочкой POSIX)

Листинг кода для регрессионных тестов (просто добавьте его в скрипт оболочки):

```
#####
# If called with 2 arguments assume they are dir paths and print rel. path #
#####

test "$#" = 2 && {
    printf '%s\n' "Rel. path from '$1' to '$2' is '$(relpath "$1" "$2")'."
    exit 0
}

#####
# If NOT called with 2 arguments run regression tests #
#####

format="\t%-19s %-22s %-27s %-8s %-8s %-8s\n"
printf \
"\n\n*** Testing own and python's function with canonical absolute dirs\n\n"
printf "$format\n" \
    "From Directory" "To Directory" "Rel. Path" "relPath" "relpath" "python"
IFS=
while read -r p; do
    eval set -- $p
    case $1 in '#'*) continue;; esac # Skip comments and empty lines
    # q stores quoting character, use " if ' is used in path name
    q="''"; case $1$2 in *'') q="''";; esac
    rPOk=passed rP=$(relPath "$1" "$2"); test "$rP" = "$3" || rPOk=$rP
    rpOk=passed rp=$(relpath "$1" "$2"); test "$rp" = "$3" || rpOk=$rp
    RPOk=passed
    RP=$(python -c "import os.path; print os.path.relpath($q$2$q, $q$1$q)")
    test "$RP" = "$3" || RPOk=$RP
    printf \
"$format" "$q$1$q" "$q$2$q" "$q$3$q" "$q$rPOk$q" "$q$rpOk$q" "$q$RPOk$q"
done <<-"EOF"

# From directory      To directory      Expected relative path

'/'                  '/'                  '.'
'/usr'               '/'                  '..'
'/usr/'              '/'                  '..'
'/usr'               '/usr'              'usr'
'/usr/'              '/usr/'              'usr'
'/usr'               '/usr'              '.'
'/usr/'              '/usr'              '.'
'/usr'               '/usr/'              '.'
'/usr/'              '/usr/'              '.'
'/u'                 '/usr'              '../usr'
'/usr'               '/u'                '../u'
"/u'/dir"            "/u'/dir"            "."
"/u'"                "/u'/dir"            "dir"
"/u'/dir"            "/u'"               ".."
"/"                  "/u'/dir"            "u'/dir"
"/u'/dir"            "/"                  "../.."
"/u'"                "/u'"               "."
"/"                  "/u'"               "u'"
"/u'"                "/"                  ".."
'/u"/dir'            '/u"/dir'            '.'
'/u"'                '/u"/dir'            'dir'
'/u"/dir'            '/u"'               '..'
```

' / - '	' / - '	' . '
' / ? '	' / ? '	' . '
' / ? ? '	' / ? ? '	' . '
' / ? ? ? '	' / ? ? ? '	' . '
' / ? * '	' / ? * '	' . '
' / * '	' / * '	' . '
' / * '	' / * * '	' . . / * * '
' / * '	' / * * * '	' . . / * * * '
' / * * * '	' / * . * * '	' . . / * . * * '

```

'/*.???'      '/*.???'      '../*.*??'
'/'          '/'          '.'
'/[a-z]*'     '/[0-9]*'     '../[0-9]*'
EOF

format="\t%-19s %-22s %-27s %-8s %-8s\n"
printf "\n\n*** Testing own and python's function with arbitrary dirs\n\n"
printf "$format\n" \
    "From Directory" "To Directory" "Rel. Path" "relpath" "python"
IFS=
while read -r p; do
    eval set -- $p
    case $1 in '#'|'|') continue;; esac # Skip comments and empty lines
    # q stores quoting character, use " if ' is used in path name
    q=""; case $1$2 in *'"'* ) q="";; esac
    rpOk=passed rp=$(relpath "$1" "$2"); test "$rp" = "$3" || rpOk=$rp
    RPOk=passed
    RP=$(python -c "import os.path; print os.path.relpath($q$2$q, $q$1$q)")
    test "$RP" = "$3" || RPOk=$RP
    printf "$format" "$q$1$q" "$q$2$q" "$q$3$q" "$q$rpOk$q" "$q$RPOk$q"
done <<-EOF"
# From directory      To directory      Expected relative path

'usr/p1/../../../../p4'  'p3/..p1/p6/../../../../p2'  '.....p1/p2'
'./home/../../../../work'  '...../dir/'      '...../dir'

'home/p1/p2'      'home/p1/p3'      './p3'
'home/p1/p2'      'home/p4/p5'      '.....p4/p5'
'home/p1/p2'      'work/p6/p7'      '...../work/p6/p7'
'home/p1'         'work/p1/p2/p3/p4'  '...../work/p1/p2/p3/p4'
'home'           'work/p2/p3'       './work/p2/p3'
'.'             'work/p2/p3'       'work/p2/p3'
'home/p1/p2'     'home/p1/p2/p3/p4'  'p3/p4'
'home/p1/p2'     'home/p1/p2/p3'     'p3'
'home/p1/p2'     'home/p1/p2'        '.'
'home/p1/p2'     'home/p1'           '..'
'home/p1/p2'     'home'              '...'
'home/p1/p2'     '.'                 '.....'
'home/p1/p2'     'work'              '...../work'
'home/p1/p2'     'work/p1'           '...../work/p1'
'home/p1/p2'     'work/p1/p2'        '...../work/p1/p2'
'home/p1/p2'     'work/p1/p2/p3'     '...../work/p1/p2/p3'
'home/p1/p2'     'work/p1/p2/p3/p4'  '...../work/p1/p2/p3/p4'
EOF

```

Делиться Улучшить этот ответ

Следовать

отредактировано 23 мая 2017 г.
в 10:31

ответил 19 сен 2013 в 15:24



Сообщество Бот

1 1



linuxball

187 1 4

Не многие из ответов здесь практичны для повседневного использования.

Поскольку очень сложно сделать это правильно в чистом bash, я предлагаю

12

следующее надежное решение (похожее на одно предложение, зарытое в комментарии):

```

function relpath() {
    python -c "import os,sys;print(os.path.relpath(*sys.argv[1:]))" "$@";

```



}

Затем вы можете получить относительный путь на основе текущего каталога:

```
echo $(relpath somepath)
```

или вы можете указать, что путь должен быть относительным к заданному каталогу:

```
echo $(relpath somepath /etc) # relative to /etc
```

Единственный недостаток – для этого требуется Python, но:


- Он работает одинаково в любом Python ≥ 2.6 .
- Существование файлов или каталогов не требуется.
- Имена файлов могут содержать более широкий диапазон специальных символов. Например, многие другие решения не работают, если имена файлов содержат пробелы или другие специальные символы.
- Это однострочная функция, которая не загромождает скрипты.

Обратите внимание, что решения, которые включают `basename` или `dirname` не обязательно могут быть лучше, так как они требуют `coreutils` установки. Если у кого-то есть чистое `bash` решение, которое надежно и просто (а не замысловатое любопытство), я буду удивлен.

Делиться Улучшить этот ответ
Следовать

отредактировано 29 июля 2015 г.
в 7:09

ответил 6 июля 2015 г. в 1:09

 Ав Гари Вишневски
польз: 1,110 10 9

Это, по-видимому, самый надежный подход. – [dimo414](#) 25 апр. 2020 г. в 10:41



8



Этот скрипт дает правильные результаты только для входных данных, которые являются абсолютными путями или относительными путями без `.` или `..`:

```
#!/bin/bash

# usage: relpath from to

if [[ "$1" == "$2" ]]
then
    echo "."
    exit
fi

IFS="/"

current=($1)
```

```

absolute=($2)

abssize=${#absolute[@]}
cursize=${#current[@]}

while [[ ${absolute[level]} == ${current[level]} ]]
do
    (( level++ ))
    if (( level > abssize || level > cursize ))
    then
        break
    fi
done

for ((i = level; i < cursize; i++))
do
    if ((i > level))
    then
        newpath=$newpath"/"
    fi
    newpath=$newpath".."
done

for ((i = level; i < abssize; i++))
do
    if [[ -n $newpath ]]
    then
        newpath=$newpath"/"
    fi
    newpath=$newpath${absolute[i]}
done

echo "$newpath"

```

Делиться Улучшить этот ответ

отредактировано 24 июля 2011 г. в 1:19

ответил 2 апр. 2010 г. в 5:17

Следовать



Деннис Уильямсон

358 тыс. 93 379 442

- 1 Кажется, это работает. Если каталоги действительно существуют, использование `$(readlink -f $1)` и `$(readlink -f $2)` на входах может исправить проблему, когда "." или ".." появляются на входах. Это может вызвать некоторые проблемы, если каталоги на самом деле не существуют. – [ТАК воняло](#) 23 июля 2011 г., 13:30



7



Я бы просто использовал Perl для этой не такой уж и тривиальной задачи:

```

absolute="/foo/bar"
current="/foo/baz/foo"

# Perl is magic
relative=$(perl -MFile::Spec -e 'print File::Spec->abs2rel("'"$absolute"'",'"$current"'")')

```

Делиться Улучшить этот ответ Следовать

ответил 12 февр. 2012 г. в 17:08

пользователь1205347

- 1 +1, но рекомендовал бы: `perl -MFile::Spec -e "print File::Spec->abs2rel('$absolute','$current')"` указывать абсолютные и текущие данные.
- Уильям Перселл 17 авг. 2014 в 13:41
- 1 Мне нравится `relative=$(perl -MFile::Spec -e 'print File::Spec->abs2rel(@ARGV)' '$absolute' '$current')` . Это гарантирует, что значения сами по себе не могут содержать код Perl! - Эрик Аронести 9 сен 2015 в 16:46



Еще одно решение, чистое `bash` + GNU `readlink` для простого использования в следующем контексте:

7



```
ln -s "$(relpath "$A" "$B")" "$B"
```



Редактировать: Убедитесь, что «\$B» либо не существует, либо в этом случае нет софтлинка, в противном случае `relpath` перейдите по этой ссылке, что не то, что вам нужно!

Это работает почти во всех текущих Linux. Если `readlink -m` у вас не работает, попробуйте `readlink -f` вместо этого. См. также <https://gist.github.com/hilbix/1ec361d00a8178ae8ea0> для возможных обновлений:

```
: relpath A B
# Calculate relative path from A to B, returns true on success
# Example: ln -s "$(relpath "$A" "$B")" "$B"
relpath()
{
    local X Y A
    # We can create dangling softlinks
    X="$(readlink -m -- "$1")" || return
    Y="$(readlink -m -- "$2")" || return
    X="${X%/}/"
    A=""
    while Y="${Y%/*}"
    [ ".$X#"$Y"/" = ".$X" ]
    do
        A="../$A"
    done
    X="$A${X#"$Y"/}"
    X="${X%/}"
    echo "${X:-.}"
}
```

Примечания:

- Были приняты меры по обеспечению безопасности от нежелательного расширения метасимволов оболочки в случае, если имена файлов содержат `*` или `?`.
- Вывод предназначен для использования в качестве первого аргумента `ln -s` :
 - `relpath /` дает `.` и не пустая строка

- `relpath a a` дает `a`, даже если `a` это каталог
- Наиболее распространенные случаи также были проверены, чтобы получить разумные результаты.
- Это решение использует сопоставление префиксов строк, поэтому `readlink` требуется канонизировать пути.
- Благодаря этому `readlink -m` он работает и для еще не существующих путей.

На старых системах, где `readlink -m` недоступно, `readlink -f` происходит сбой, если файл не существует. Так что вам, вероятно, понадобится обходной путь вроде этого (непроверено!):

```
readlink_missing()
{
  readlink -m -- "$1" && return
  readlink -f -- "$1" && return
  [ -e . ] && echo "$(readlink_missing "$(dirname "$1")")/$(basename "$1")"
}
```

Это не совсем верно в случае `$1` включений `.` или `..` несуществующих путей (как в `/doesnotexist/./a`), но должно охватывать большинство случаев.

(Заменить `readlink -m --` выше на `readlink_missing .`)

Изменить из-за отрицательного голоса следует

Вот тест, который подтверждает, что эта функция действительно верна:

```
check()
{
  res="$(relpath "$2" "$1")"
  [ "$res" = ".$3" ] && return
  printf ':WRONG: %-10q %-10q gives %q\nCORRECT %-10q %-10q gives %q\n' "$1" "$2"
    "$res" "$@"
}
```

#	TARGET	SOURCE	RESULT
check	<code>"/A/B/C"</code>	<code>"/A"</code>	<code>".."</code>
check	<code>"/A/B/C"</code>	<code>"/A.x"</code>	<code>"../..A.x"</code>
check	<code>"/A/B/C"</code>	<code>"/A/B"</code>	<code>"."</code>
check	<code>"/A/B/C"</code>	<code>"/A/B/C"</code>	<code>"C"</code>
check	<code>"/A/B/C"</code>	<code>"/A/B/C/D"</code>	<code>"C/D"</code>
check	<code>"/A/B/C"</code>	<code>"/A/B/C/D/E"</code>	<code>"C/D/E"</code>
check	<code>"/A/B/C"</code>	<code>"/A/B/D"</code>	<code>"D"</code>
check	<code>"/A/B/C"</code>	<code>"/A/B/D/E"</code>	<code>"D/E"</code>
check	<code>"/A/B/C"</code>	<code>"/A/D"</code>	<code>"../D"</code>
check	<code>"/A/B/C"</code>	<code>"/A/D/E"</code>	<code>"../D/E"</code>
check	<code>"/A/B/C"</code>	<code>"/D/E/F"</code>	<code>"../..D/E/F"</code>

```
check "/foo/baz/moo" "/foo/bar" "../bar"
```

Озадачены? Ну, это правильные результаты ! Даже если вы думаете, что это не соответствует вопросу, вот доказательство того, что это правильно:

```
check "http://example.com/foo/baz/moo" "http://example.com/foo/bar" "../bar"
```

Без сомнения, `../bar` это точный и единственно правильный относительный путь страницы, `bar` видимый со страницы `moo`. Все остальное было бы просто неправильным.

Тривиально адаптировать вывод к вопросу, который, по-видимому, предполагает, что `current` это каталог:

```
absolute="/foo/bar"
current="/foo/baz/foo"
relative="../$(relpath "$absolute" "$current")"
```

Это возвращает именно то, что было запрошено.

И прежде чем вы удивитесь, вот немного более сложный вариант `relpath` (найдите небольшое отличие), который должен работать и для URL-синтаксиса (так что конечный символ `/` сохранится, благодаря некоторой `bash`-магии):

```
# Calculate relative PATH to the given DEST from the given BASE
# In the URL case, both URLs must be absolute and have the same Scheme.
# The `SCHEME:` must not be present in the FS either.
# This way this routine works for file paths an
: relpathurl DEST BASE
relpathurl()
{
    local X Y A
    # We can create dangling softlinks
    X="$(readlink -m -- "$1")" || return
    Y="$(readlink -m -- "$2")" || return
    X="${X%/}/${1#"${1%}/"}"
    Y="${Y%/}/${2#"${2%}/"}"
    A=""
    while Y="${Y%/*}"
    [ ".${X#"${X%}/"}" = ".$X" ]
    do
        A="../$A"
    done
    X="$A${X#"${X%}/"}"
    X="${X%/}"
    echo "${X:-.}"
}
```

И вот проверки, чтобы прояснить: все действительно работает так, как и заявлено.

```
check()
{
    res="$(relpathurl "$2" "$1")"
    [ ".$res" = ".$3" ] && return
    printf ':WRONG: %-10q %-10q gives %q\nCORRECT %-10q %-10q gives %q\n' "$1" "$2"
    "$res" "$@"
}

#      TARGET      SOURCE      RESULT
```

```

check "/A/B/C" "/A"           ".."
check "/A/B/C" "/A.x"        "../..A.x"
check "/A/B/C" "/A/B"        "."
check "/A/B/C" "/A/B/C"       "C"
check "/A/B/C" "/A/B/C/D"     "C/D"
check "/A/B/C" "/A/B/C/D/E"   "C/D/E"
check "/A/B/C" "/A/B/D"       "D"
check "/A/B/C" "/A/B/D/E"     "D/E"
check "/A/B/C" "/A/D"         "../D"
check "/A/B/C" "/A/D/E"       "../D/E"
check "/A/B/C" "/D/E/F"       "../..D/E/F"

check "/foo/baz/moo" "/foo/bar" "../bar"
check "http://example.com/foo/baz/moo" "http://example.com/foo/bar" "../bar"

check "http://example.com/foo/baz/moo/" "http://example.com/foo/bar" "../..bar"
check "http://example.com/foo/baz/moo" "http://example.com/foo/bar/" "../bar/"
check "http://example.com/foo/baz/moo/" "http://example.com/foo/bar/"
"../..bar/"

```

И вот как это можно использовать, чтобы получить желаемый результат из вопроса:

```

absolute="/foo/bar"
current="/foo/baz/foo"
relative="$(relpathurl "$absolute" "$current/")"
echo "$relative"

```

Если вы обнаружите что-то, что не работает, пожалуйста, дайте мне знать в комментариях ниже. Спасибо.

P.S.:

Почему аргументы `relpath` «перевернуты» в отличие от всех остальных ответов здесь?

Если вы измените

```
Y="$(readlink -m -- "$2")" || return
```

к

```
Y="$(readlink -m -- "${2:-"$PWD"}")" || return
```

тогда вы можете оставить 2-й параметр, так что `BASE` будет текущим каталогом/URL/чем угодно. Это всего лишь принцип Unix, как обычно.

Делиться Улучшить этот ответ
Следовать

отредактировано 31 окт. 2020 г.
в 19:48



Асклепий

62.4k

19

184

155

ответил 27 нояб. 2014 г. в
16:58



Абсолютно

10,2 тыс.

5

56

62



Небольшое улучшение ответов [kasku](#) и [Pini](#) , которое лучше работает с пробелами и позволяет передавать относительные пути:

6



```
#!/bin/bash
# both $1 and $2 are paths
# returns $2 relative to $1
absolute=`readlink -f "$2"`
current=`readlink -f "$1"`
# Perl is magic
# Quoting horror.... spaces cause problems, that's why we need the extra " in
here:
relative=$(perl -MFile::Spec -e "print File::Spec-
>abs2rel(q($absolute),q($current))")

echo $relative
```



Делиться Улучшить этот ответ

Следовать

отредактировано 23 мая 2017 г.
в 12:26

сообщество вики
5 оборотов, 2 пользователя
87%
синусоидальный



К сожалению, ответ Марка Рашакова (теперь удаленный – он ссылался на код отсюда [\](#) , похоже, работает некорректно, если адаптировать его к:

4



```
source=/home/part2/part3/part4
target=/work/proj1/proj2
```



Мысль, изложенная в комментарии, может быть улучшена, чтобы она работала правильно в большинстве случаев. Я собираюсь предположить, что скрипт принимает аргумент источника (где вы находитесь) и аргумент цели (куда вы хотите попасть), и что либо оба являются абсолютными путями, либо оба являются относительными. Если один из них абсолютный, а другой относительный, проще всего добавить к относительному имени префикс текущего рабочего каталога, но код ниже этого не делает.

Остерегаться

Приведенный ниже код близок к корректной работе, но не совсем верен.

1. Эту проблему поднимает в своих комментариях Деннис Уильямсон.
2. Проблема также в том, что это чисто текстовая обработка имен путей, и вы можете серьезно запутаться из-за странных символических ссылок.
3. Код не обрабатывает случайные «точки» в путях, такие как « `xyz/./pqr` ».
4. Код не обрабатывает случайные «двойные точки» в путях типа « `xyz/../pqr` ».
5. Тривиально: код не удаляет начальные символы ' `./` ' из путей.

Код Денниса лучше, потому что он исправляет ошибки 1 и 5, но имеет те же проблемы 2, 3, 4. Используйте код Денниса (и голосуйте за него заранее) именно по этой причине.

(Примечание: POSIX предоставляет системный вызов `realpath()`, который разрешает имена путей так, чтобы в них не оставалось символических ссылок. Применение этого к именам входных данных и последующее использование кода Денниса каждый раз давали бы правильный ответ. Написать код C, который выполняет обертку, несложно `realpath()` — я это делал, — но я не знаю стандартной утилиты, которая это делает.)

Для этого я нахожу Perl более удобным, чем shell, хотя bash имеет достойную поддержку массивов и, вероятно, мог бы делать это тоже — упражнение для читателя. Итак, имея два совместимых имени, разделим каждое из них на компоненты:

- Установите пустой относительный путь.
- Поскольку компоненты те же самые, переходите к следующему.
- Когда соответствующие компоненты различны или для одного пути больше нет компонентов:
- Если исходных компонентов не осталось и относительный путь пуст, добавьте «.» в начало.
- Для каждого оставшегося исходного компонента добавьте к относительному пути префикс «../».
- Если целевых компонентов не осталось и относительный путь пуст, добавьте «.» в начало.
- Для каждого оставшегося целевого компонента добавьте компонент в конец пути после косой черты.

Таким образом:

```
#!/bin/perl -w

use strict;

# Should fettle the arguments if one is absolute and one relative:
# Oops - missing functionality!

# Split!
my(@source) = split '/', $ARGV[0];
my(@target) = split '/', $ARGV[1];

my $count = scalar(@source);
$count = scalar(@target) if (scalar(@target) < $count);
my $relpath = "";

my $i;
for ($i = 0; $i < $count; $i++)
{
    last if $source[$i] ne $target[$i];
}
```

```

}

$relpath = "." if ($i >= scalar(@source) && $relpath eq "");
for (my $s = $i; $s < scalar(@source); $s++)
{
    $relpath = "../$relpath";
}
$relpath = "." if ($i >= scalar(@target) && $relpath eq "");
for (my $t = $i; $t < scalar(@target); $t++)
{
    $relpath .= "$target[$t]";
}

# Clean up result (remove double slash, trailing slash, trailing slash-dot).
$relpath =~ s%/%%/%;
$relpath =~ s%/$%%;
$relpath =~ s%/\. $%%;

print "source  = $ARGV[0]\n";
print "target  = $ARGV[1]\n";
print "relpath = $relpath\n";

```

Тестовый сценарий (в квадратных скобках пробел и табуляция):

```

sed 's/#.*//;/^[ ]*$$/d' <<! |

/home/part1/part2 /home/part1/part3
/home/part1/part2 /home/part4/part5
/home/part1/part2 /work/part6/part7
/home/part1      /work/part1/part2/part3/part4
/home            /work/part2/part3
/                /work/part2/part3/part4

/home/part1/part2 /home/part1/part2/part3/part4
/home/part1/part2 /home/part1/part2/part3
/home/part1/part2 /home/part1/part2
/home/part1/part2 /home/part1
/home/part1/part2 /home
/home/part1/part2 /

/home/part1/part2 /work
/home/part1/part2 /work/part1
/home/part1/part2 /work/part1/part2
/home/part1/part2 /work/part1/part2/part3
/home/part1/part2 /work/part1/part2/part3/part4

home/part1/part2 home/part1/part3
home/part1/part2 home/part4/part5
home/part1/part2 work/part6/part7
home/part1      work/part1/part2/part3/part4
home            work/part2/part3
.                work/part2/part3

home/part1/part2 home/part1/part2/part3/part4
home/part1/part2 home/part1/part2/part3
home/part1/part2 home/part1/part2
home/part1/part2 home/part1
home/part1/part2 home
home/part1/part2 .

home/part1/part2 work
home/part1/part2 work/part1
home/part1/part2 work/part1/part2
home/part1/part2 work/part1/part2/part3

```

```
home/part1/part2 work/part1/part2/part3/part4
```

```
!
```

```
while read source target
do
    perl relpath.pl $source $target
    echo
done
```

Вывод тестового скрипта:

```
source = /home/part1/part2
target = /home/part1/part3
relpath = ../part3

source = /home/part1/part2
target = /home/part4/part5
relpath = ../../part4/part5

source = /home/part1/part2
target = /work/part6/part7
relpath = ../../../../work/part6/part7

source = /home/part1
target = /work/part1/part2/part3/part4
relpath = ../../work/part1/part2/part3/part4

source = /home
target = /work/part2/part3
relpath = ../work/part2/part3

source = /
target = /work/part2/part3/part4
relpath = ./work/part2/part3/part4

source = /home/part1/part2
target = /home/part1/part2/part3/part4
relpath = ./part3/part4

source = /home/part1/part2
target = /home/part1/part2/part3
relpath = ./part3

source = /home/part1/part2
target = /home/part1/part2
relpath = .

source = /home/part1/part2
target = /home/part1
relpath = ..

source = /home/part1/part2
target = /home
relpath = ../../

source = /home/part1/part2
target = /
relpath = ../../../../

source = /home/part1/part2
target = /work
relpath = ../../../../work
```



```
source = /home/part1/part2
target = /work/part1
relpath = ../../../work/part1

source = /home/part1/part2
target = /work/part1/part2
relpath = ../../../work/part1/part2

source = /home/part1/part2
target = /work/part1/part2/part3
relpath = ../../../work/part1/part2/part3

source = /home/part1/part2
target = /work/part1/part2/part3/part4
relpath = ../../../work/part1/part2/part3/part4

source = home/part1/part2
target = home/part1/part3
relpath = ../part3

source = home/part1/part2
target = home/part4/part5
relpath = ../../part4/part5

source = home/part1/part2
target = work/part6/part7
relpath = ../../../work/part6/part7

source = home/part1
target = work/part1/part2/part3/part4
relpath = ../../work/part1/part2/part3/part4

source = home
target = work/part2/part3
relpath = ../work/part2/part3

source = .
target = work/part2/part3
relpath = ../work/part2/part3

source = home/part1/part2
target = home/part1/part2/part3/part4
relpath = ./part3/part4

source = home/part1/part2
target = home/part1/part2/part3
relpath = ./part3

source = home/part1/part2
target = home/part1/part2
relpath = .

source = home/part1/part2
target = home/part1
relpath = ..

source = home/part1/part2
target = home
relpath = ../../

source = home/part1/part2
target = .
relpath = ../../..

source = home/part1/part2
target = work
```

```

relpath = ../../../../work

source = home/part1/part2
target = work/part1
relpath = ../../../../work/part1

source = home/part1/part2
target = work/part1/part2
relpath = ../../../../work/part1/part2

source = home/part1/part2
target = work/part1/part2/part3
relpath = ../../../../work/part1/part2/part3

source = home/part1/part2
target = work/part1/part2/part3/part4
relpath = ../../../../work/part1/part2/part3/part4

```

Этот скрипт Perl работает довольно тщательно в Unix (он не учитывает все сложности имен путей Windows) при наличии странных входных данных. Он использует модуль `Cwd` и его функцию `realpath` для разрешения реального пути имен, которые существуют, и выполняет текстовый анализ для путей, которые не существуют. Во всех случаях, кроме одного, он выдает тот же вывод, что и скрипт Денниса. Отклоняющийся случай:

```

source = home/part1/part2
target = .
relpath1 = ../../..
relpath2 = ../../../../

```

Два результата эквивалентны – просто не идентичны. (Вывод получен из слегка измененной версии тестового скрипта – скрипт Perl ниже просто выводит ответ, а не входные данные и ответ, как в скрипте выше.) *Теперь: следует ли мне удалить неработающий ответ? Может быть...*

```

#!/bin/perl -w
# Based loosely on code from:
http://unix.derkeiler.com/Newsgroups/comp.unix.shell/2005-10/1256.html
# Via: http://stackoverflow.com/questions/2564634

use strict;

die "Usage: $0 from to\n" if scalar @ARGV != 2;

use Cwd qw(realpath getcwd);

my $pwd;
my $verbose = 0;

# Fettle filename so it is absolute.
# Deals with '//', './' and '/../' notations, plus symlinks.
# The realpath() function does the hard work if the path exists.
# For non-existent paths, the code does a purely textual hack.
sub resolve
{
    my($name) = @_;
    my($path) = realpath($name);

```

```

if (!defined $path)
{
    # Path does not exist - do the best we can with lexical analysis
    # Assume Unix - not dealing with Windows.
    $path = $name;
    if ($name !~ m%^/)
    {
        $pwd = getcwd if !defined $pwd;
        $path = "$pwd/$path";
    }
    $path =~ s%//+%/g;      # Not UNC paths.
    $path =~ s%/$%%;       # No trailing /
    $path =~ s%/\./%/g;    # No embedded ./
    # Try to eliminate ../abc/
    $path =~ s%/\.\./(?:[^\./]+)(/|$)%$1%g;
    $path =~ s%/\.$%%;     # No trailing /.
    $path =~ s%^\.%/g;     # No leading ./
    # What happens with . and / as inputs?
}
return($path);
}

sub print_result
{
    my($source, $target, $relpath) = @_ ;
    if ($verbose)
    {
        print "source  = $ARGV[0]\n";
        print "target  = $ARGV[1]\n";
        print "relpath = $relpath\n";
    }
    else
    {
        print "$relpath\n";
    }
    exit 0;
}

my($source) = resolve($ARGV[0]);
my($target) = resolve($ARGV[1]);
print_result($source, $target, ".") if ($source eq $target);

# Split!
my(@source) = split '/', $source;
my(@target) = split '/', $target;

my $count = scalar(@source);
$count = scalar(@target) if (scalar(@target) < $count);
my $relpath = "";
my $i;

# Both paths are absolute; Perl splits an empty field 0.
for ($i = 1; $i < $count; $i++)
{
    last if $source[$i] ne $target[$i];
}

for (my $s = $i; $s < scalar(@source); $s++)
{
    $relpath = "$relpath/" if ($s > $i);
    $relpath = "$relpath..";
}
for (my $t = $i; $t < scalar(@target); $t++)
{
    $relpath = "$relpath/" if ($relpath ne "");
    $relpath = "$relpath$target[$t]";
}

```

```
}

print_result($source, $target, $relpath);
```

Делиться Улучшить этот ответ
Следовать

отредактировано 4 апр. 2010 г.
в 3:22

ответил 2 апр. 2010 г. в 4:50



Джонатан Леффлер

749k 144 943 1,3
тыс.

Ваш `/home/part1/part2 to /` имеет слишком много `../`. В противном случае мой скрипт соответствует вашему выводу, за исключением того, что мой добавляет ненужный `.` в конце того, где находится пункт назначения, `.` и я не использую `./` в начале тех, которые спускаются без подъема. – [Деннис Уильямсон](#) 2 апреля 2010 г. в 5:40

@Dennis: Я потратил время, косясь на результаты – иногда я мог видеть эту проблему, а иногда не мог ее снова найти. Удаление начального `'./'` – еще один тривиальный шаг. Ваш комментарий о 'без встроенных `.` или `../`' также уместен. На самом деле удивительно сложно выполнить работу правильно – вдвойне сложно, если какое-либо из имен на самом деле является символической ссылкой; мы оба делаем чисто текстовый анализ.
– [Джонатан Леффлер](#) 2 апреля 2010 г. в 6:28

@Dennis: Конечно, если у вас нет сети Newcastle Connection, попытки получить права выше `root` бесполезны, поэтому `../../../../` и `../../../../` эквивалентны. Однако это чистый эскапизм; ваша критика верна. (Newcastle Connection позволяла вам настраивать и использовать обозначение `./host/path/on/remote/machine` для перехода на другой хост – изящная схема. Я думаю, что она также поддерживала `../../../../network/host/path/on/remote/network/and/host`. Это есть в Википедии.)
– [Джонатан Леффлер](#) 2 апреля 2010 г. в 6:31

Поэтому вместо этого у нас теперь двойной слеш `UNC`. – [Деннис Уильямсон](#) 2 апр. 2010 г. в 10:10

- 2 Утилита "readlink" (по крайней мере, версия GNU) может выполнять эквивалент `realpath()`, если вы передадите ей опцию `"-f"`. Например, в моей системе, `readlink /usr/bin/vi` returns `/etc/alternatives/vi`, но это еще одна символическая ссылка - тогда как `readlink -f /usr/bin/vi` returns `/usr/bin/vim.basic`, которая является конечным пунктом назначения всех символических ссылок... – [psmeares](#) 14 мая 2010 г., 9:54 ✎



тест.ш:

4

```
#!/bin/bash
```



```
cd /home/ubuntu
touch blah
TEST=/home/ubuntu/./blah
echo TEST=$TEST
TMP=$(readlink -e "$TEST")
echo TMP=$TMP
REL=${TMP##$(pwd)/}
echo REL=$REL
```



Тестирование:

```
$ ./test.sh
TEST=/home/ubuntu/./blah
TMP=/home/ubuntu/blah
REL=blah
```

Делиться Улучшить этот ответ Следовать

ответил 11 июня 2010 г. в 17:07



Стив
73 1

+1 за компактность и баш-ность. Однако вам следует также обратиться `readlink` к `$(pwd)` . - [DevSolar](#) 29 ноя 2010 в 16:06

2 Относительность не означает, что файл должен быть помещен в тот же каталог. - [зеленыйолдман](#) 26 марта 2011 г. в 15:47

Хотя в исходном вопросе не приводится много тестовых случаев, этот скрипт не справляется с простыми тестами, такими как поиск относительного пути от `/home/user1` до `/home/user2` (правильный ответ: `../user2`). Скрипт от `pinijswenger` подходит для этого случая. - [Майкл](#) 28 мар. 2012 г. в 22:15



3



Я воспринял ваш вопрос как вызов написать это в «переносимом» шелл-коде, т.е.

- с учетом оболочки POSIX
- нет башизмов, таких как массивы
- избегайте вызова внешних как чумы. В скрипте нет ни одного форка! Это делает его невероятно быстрым, особенно на системах со значительными накладными расходами на форк, таких как `cygwin`.
- Необходимо иметь дело с символами `glob` в именах путей (`*`, `?`, `[`, `]`)

Он работает на любой оболочке, соответствующей POSIX (`zsh`, `bash`, `ksh`, `ash`, `busybox`, ...). Он даже содержит набор тестов для проверки своей работы. Канонизация имен путей оставлена в качестве упражнения. :-)

```
#!/bin/sh

# Find common parent directory path for a pair of paths.
# Call with two pathnames as args, e.g.
# commondirpart foo/bar foo/baz/bat -> result="foo/"
# The result is either empty or ends with "/".
commondirpart () {
    result=""
    while test ${#1} -gt 0 -a ${#2} -gt 0; do
        if test "${1%${1#?}}" != "${2%${2#?}}"; then # First characters the same?
            break # No, we're done comparing.
        fi
        result="$result${1%${1#?}}" # Yes, append to result.
        set -- "${1%?}" "${2%?}" # Chop first char off both
    done
    strings.
    case "$result" in
        ("|*") ;;
        (*) result="${result%*/}"/;;
    esac
```

[illegible]

```

/*.* /*.*.* ../*.**
/*.*?? /*.*?? ../*.??
/[] /[] .
/[a-z]* /[0-9]* ../[0-9]*
/foo /foo .
/foo / ..
/foo/bar / ../..
/foo/bar /foo ..
/foo/bar /foo/baz ../baz
/foo/bar /bar/foo ../../bar/foo
/foo/bar/baz /gnarf/blurfl/blubb ../../../gnarf/blurfl/blubb
/foo/bar/baz /gnarf ../../../gnarf
/foo/bar/baz /foo/baz ../../baz
/foo. /bar. ../bar.
EOF
while read FROM TO VIA; do
    relativepath "$FROM" "$TO"
    printf '%s\n' "FROM: $FROM" "TO: $TO" "VIA: $result"
    if test "$result" != "$VIA"; then
        printf '%s\n' "OOOPS! Expected '$VIA' but got '$result'"
    fi
done

# vi: set tabstop=3 shiftwidth=3 expandtab fileformat=unix :

```

Делиться Улучшить этот ответ

отредактировано 11 авг. 2011 г.
в 11:42

ответил 11 авг. 2011 г. в 9:16

Следовать



Йенс

72 тыс. 16 130 183

1 Это впечатляет! – говорящий с тенью 29 июл. 2022 в 16:26

Мое решение:

2

```

computeRelativePath()
{
    Source=$(readlink -f ${1})
    Target=$(readlink -f ${2})

    local OLDIFS=$IFS
    IFS="/"

    local SourceDirectoryArray=($Source)
    local TargetDirectoryArray=($Target)

    local SourceArrayLength=$(echo ${SourceDirectoryArray[@]} | wc -w)
    local TargetArrayLength=$(echo ${TargetDirectoryArray[@]} | wc -w)

    local Length
    test $SourceArrayLength -gt $TargetArrayLength && Length=$SourceArrayLength ||
    Length=$TargetArrayLength

    local Result=""
    local AppendToEnd=""

    IFS=$OLDIFS

    local i

```

```

for ((i = 0; i <= $Length + 1 ; i++ ))
do
    if [ "${SourceDirectoryArray[$i]}" = "${TargetDirectoryArray[$i]}" ]
    then
        continue
    elif [ "${SourceDirectoryArray[$i]}" != "" ] && [
"${TargetDirectoryArray[$i]}" != "" ]
    then
        AppendToEnd="${AppendToEnd}${TargetDirectoryArray[$i]}/"
        Result="${Result}../"

    elif [ "${SourceDirectoryArray[$i]}" = "" ]
    then
        Result="${Result}${TargetDirectoryArray[$i]}/"
    else
        Result="${Result}../"
    fi
done

Result="${Result}${AppendToEnd}"

echo $Result
}

```

Делиться Улучшить этот ответ Следовать

ответил 25 окт. 2011 г. в 16:37



Анонимный

1550 4 18 30

Это очень портативно :) - [Анонимный](#) 31 окт. 2011 г. в 18:30



2



Этот скрипт работает только с именами путей. Он не требует существования каких-либо файлов. Если переданные пути не абсолютны, поведение немного необычно, но он должен работать так, как и ожидалось, если оба пути относительные.

Я тестировал его только на OS X, поэтому он может оказаться непортативен.



```

#!/bin/bash
set -e
declare SCRIPT_NAME="$(basename $0)"
function usage {
    echo "Usage: $SCRIPT_NAME <base path> <target file>"
    echo "      Outputs <target file> relative to <base path>"
    exit 1
}

if [ $# -lt 2 ]; then usage; fi

declare base=$1
declare target=$2
declare -a base_part=()
declare -a target_part=()

#Split path elements & canonicalize
OFS="$IFS"; IFS='/'
bpl=0;

```



```

for bp in $base; do
    case "$bp" in
        ".") ;;
        "..") let "bpl=$bpl-1" ;;
        *) base_part[${bpl}]="$bp" ; let "bpl=$bpl+1" ;;
    esac
done
tpl=0;
for tp in $target; do
    case "$tp" in
        ".") ;;
        "..") let "tpl=$tpl-1" ;;
        *) target_part[${tpl}]="$tp" ; let "tpl=$tpl+1" ;;
    esac
done
IFS="$IFS"

#Count common prefix
common=0
for (( i=0 ; i<$bpl ; i++ )); do
    if [ "${base_part[$i]}" = "${target_part[$common]}" ] ; then
        let "common=$common+1"
    else
        break
    fi
done

#Compute number of directories up
let "updir=$bpl-$common" || updir=0 #if the expression is zero, 'let' fails

#trivial case (after canonical decomposition)
if [ $updir -eq 0 ]; then
    echo .
    exit
fi

#Print updirs
for (( i=0 ; i<$updir ; i++ )); do
    echo -n ../
done

#Print remaining path
for (( i=$common ; i<$tpl ; i++ )); do
    if [ $i -ne $common ]; then
        echo -n "/"
    fi
    if [ "" != "${target_part[$i]}" ] ; then
        echo -n "${target_part[$i]}"
    fi
done
#One last newline
echo

```

Делиться Улучшить этот ответ
Следовать

отредактировано 27 сен 2015 в 21:09



Питер Мортенсен

31,6 тыс. 22 109
132

ответил 6 дек. 2010 г. в 20:23



Хуанкн

2,593 2 20 25

Кроме того, код немного похож на копипаст, но мне это было нужно довольно быстро.

– Хуанкн 6 дек. 2010 г. в 20:25

Хорошо... как раз то, что мне было нужно. И вы включили процедуру канонизации, которая лучше большинства других, которые я видел (которые обычно полагаются на замены регулярных выражений). – [drwatsoncode](#) 10 окт. 2015 г. в 6:24



Вот моя версия. Она основана на [ответе](#) @Offirmo . Я сделал ее совместимой с Dash и исправил следующую ошибку тестового случая [:](#)

2



```
./compute-relative.sh "/a/b/c/de/f/g" "/a/b/c/def/g/" --> "../../../f/g/"
```

Сейчас:



```
CT_FindRelativePath "/a/b/c/de/f/g" "/a/b/c/def/g/" --> "../../../def/g/"
```



Смотрите код:

```
# both $1 and $2 are absolute paths beginning with /
# returns relative path to $2/$target from $1/$source
CT_FindRelativePath()
{
    local insource=$1
    local intarget=$2

    # Ensure both source and target end with /
    # This simplifies the inner loop.
    #echo "insource : \"$insource\""
    #echo "intarget : \"$intarget\""
    case "$insource" in
        */) ;;
        *) source="$insource"/ ;;
    esac

    case "$intarget" in
        */) ;;
        *) target="$intarget"/ ;;
    esac

    #echo "source : \"$source\""
    #echo "target : \"$target\""

    local common_part=$source # for now

    local result=""

    #echo "common_part is now : \"$common_part\""
    #echo "result is now      : \"$result\""
    #echo "target#common_part : \"${target#$common_part}\""
    while [ "${target#$common_part}" = "${target}" -a "${common_part}" != "/" ];
do
    # no match, means that candidate common part is not correct
    # go up one level (reduce common part)
    common_part=$(dirname "$common_part")/
    # and record that we went back
    if [ -z "${result}" ]; then
        result="../"
    else
        result="$../$result"
    fi
    #echo "(w) common_part is now : \"$common_part\""
    #echo "(w) result is now      : \"$result\""
    #echo "(w) target#common_part : \"${target#$common_part}\""
```

```
done

#echo "(f) common_part is      : \"${common_part}\""

if [ "${common_part}" = "/" ]; then
    # special case for root (no common path)
    common_part="/"
fi

# since we now have identified the common part,
# compute the non-common part
forward_part="${target}${common_part}"
#echo "forward_part = \"${forward_part}\""

if [ -n "${result}" -a -n "${forward_part}" ]; then
    #echo "(simple concat)"
    result="${result}${forward_part}"
elif [ -n "${forward_part}" ]; then
    result="${forward_part}"
fi
#echo "result = \"${result}\""

# if a / was added to target and result ends in / then remove it now.
if [ "${intarget}" != "$target" ]; then
    case "$result" in
        */) result=$(echo "$result" | awk '{ string=substr($0, 1,
length($0)-1); print string; }' ) ;;
    esac
fi

echo $result

return 0
}
```

Делиться Улучшить этот ответ

Следовать

отредактировано 23 мая 2017 г.
в 12:34

ответил 11 июня 2015 г. в 11:03



Сообщество Бот
1 1



Рэй Доннелли
4,006 1 20 20



2



Я использую macOS, в которой по умолчанию нет `realpath` команды, поэтому я создал `pure bash` функцию для ее вычисления.

```
#!/bin/bash

##
# print a relative path from "source folder" to "target file"
#
# params:
# $1 - target file, can be a relative path or an absolute path.
# $2 - source folder, can be a relative path or an absolute path.
#
# test:
# $ mkdir -p ~/A/B/C/D; touch ~/A/B/C/D/testfile.txt; touch ~/A/B/testfile.txt
#
# $ getRelativePath ~/A/B/C/D/testfile.txt ~/A/B
# $ C/D/testfile.txt
#
# $ getRelativePath ~/A/B/testfile.txt ~/A/B/C
# $ ../testfile.txt
#
```

```
# $ getRelativePath ~/A/B/testfile.txt /
# $ home/bunnier/A/B/testfile.txt
#
function getRelativePath() {
    local targetFilename=$(basename $1)
    local targetFolder=$(cd $(dirname $1);pwd) # absolute target folder path
    local currentFolder=$(cd $2;pwd) # absolute source folder
    local result=.

    while [ "$currentFolder" != "$targetFolder" ];do
        if [[ "$targetFolder" =~ "$currentFolder"* ]];then
            pointSegment=${targetFolder#$currentFolder}
            result=$result/${pointSegment#/}
            break
        fi
        result="$result"/..
        currentFolder=$(dirname $currentFolder)
    done

    result=$result/$targetFilename
    echo ${result#./}
}
```

Делиться Улучшить этот ответ Следовать

ответил 8 мая 2021 г. в 15:22



кролик

81 5

Думаю, этот тоже сработает... (в комплекте есть встроенные тесты) :)

1

Ладно, ожидаются некоторые накладные расходы, но мы же тут используем оболочку Bourne! ;)

```
#!/bin/sh

#
# Finding the relative path to a certain file ($2), given the absolute path ($1)
# (available here too http://pastebin.com/tWWqA8aB)
#
relpath () {
    local FROM="$1"
    local TO=`dirname $2`
    local FILE=`basename $2`
    local DEBUG="$3"

    local FROMREL=""
    local FROMUP="$FROM"
    while [ "$FROMUP" != "/" ]; do
        local TOUP="$TO"
        local TOREL=""
        while [ "$TOUP" != "/" ]; do
            [ -z "$DEBUG" ] || echo 1>&2 "DEBUG$FROMUP =?= $TOUP"
            if [ "$FROMUP" = "$TOUP" ]; then
                echo "${FROMREL:-.}/${TOREL}${TOREL:+/}$FILE"
                return 0
            fi
            TOREL=`basename $TOUP`${TOREL:+/}$TOREL`
            TOUP=`dirname $TOUP`
        done
        FROMREL="..${FROMREL:+/}$FROMREL"
        FROMUP=`dirname $FROMUP`
    done
}
```

```

done
echo "${FROMREL:-.}${TOREL:+/}${TOREL}/${FILE}"
return 0
}

relpathshow () {
    echo " - target $2"
    echo "   from   $1"
    echo "   -----"
    echo "   => `relpath $1 $2` '      ' ``"
    echo ""
}

# If given 2 arguments, do as said...
if [ -n "$2" ]; then
    relpath $1 $2

# If only one given, then assume current directory
elif [ -n "$1" ]; then
    relpath `pwd` $1

# Otherwise perform a set of built-in tests to confirm the validity of the method!
;)
else

    relpathshow /usr/share/emacs22/site-lisp/emacs-goodies-el \
                /usr/share/emacs22/site-lisp/emacs-goodies-el/filladapt.el

    relpathshow /usr/share/emacs23/site-lisp/emacs-goodies-el \
                /usr/share/emacs22/site-lisp/emacs-goodies-el/filladapt.el

    relpathshow /usr/bin \
                /usr/share/emacs22/site-lisp/emacs-goodies-el/filladapt.el

    relpathshow /usr/bin \
                /usr/share/emacs22/site-lisp/emacs-goodies-el/filladapt.el

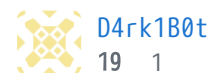
    relpathshow /usr/bin/share/emacs22/site-lisp/emacs-goodies-el \
                /etc/motd

    relpathshow / \
                /initrd.img
fi

```

Делиться Улучшить этот ответ Следовать

ответил 23 дек. 2011 г. в 23:05



0

Этот ответ не затрагивает часть вопроса, касающуюся Bash, но поскольку я попытался использовать ответы в этом вопросе для реализации этой функциональности в [Emacs](#), я опубликую ее там.



На самом деле в Emacs для этого есть встроенная функция:



```
ELISP> (file-relative-name "/a/b/c" "/a/b/c")
"."
```



```
ELISP> (file-relative-name "/a/b/c" "/a/b")
"c"
```

```
ELISP> (file-relative-name "/a/b/c" "/c/b")
"../../a/b/c"
```

Делиться Улучшить этот ответ
Следовать

отредактировано 27 сен 2015 в 21:16



Питер Мортенсен

31,6 тыс. 22 109
132

ответил 16 июня 2015 г. в 11:59



фальшивыйдракон

6,816 8 48 68

Обратите внимание, что я считаю, что ответ на Python, который я недавно добавил (функция `relpath`), ведет себя идентично `file-relative-name` представленным вами тестовым случаям. – Гари Вишневски 28 июля 2015 г., 0:10



в баше:

0



```
realDir=''
cd $(dirname $0) || exit
realDir=$(pwd)
cd -
echo $realDir
```



Делиться Улучшить этот ответ Следовать

ответил 18 нояб. 2022 г. в 8:17



джемли

1,323 1 14 28



Вот скрипт оболочки, который делает это без вызова других программ:

-1



```
#!/bin/env bash

#bash script to find the relative path between two directories

mydir=${0%/}
mydir=${0%/*}
creadlink="$mydir/creadlink"

shopt -s extglob

relpath_ () {
    path1=("$creadlink" "$1")
    path2=("$creadlink" "$2")
    orig1=$path1
    path1=${path1%/}/
    path2=${path2%/}/

    while ;; do
        if test ! "$path1"; then
            break
        fi
        part1=${path2#$path1}
        if test "${part1#/}" = "$part1"; then
            path1=${path1%/*}
            continue
        fi
        if test "${path2#$path1}" = "$path2"; then
```

```

        path1=${path1%/*}
        continue
    fi
    break
done
part1=$path1
path1=${orig1#$part1}
depth=${path1//+([^\/] )/..}
path1=${path2#$path1}
path1=${depth}${path2#$part1}
path1=${path1##+([^\/] )}
path1=${path1%/}
if test ! "$path1"; then
    path1=.
fi
printf "$path1"
}

relpath_test () {
    res=$(relpath_ /path1/to/dir1 /path1/to/dir2 )
    expected='../dir2'
    test_results "$res" "$expected"

    res=$(relpath_ / /path1/to/dir2 )
    expected='path1/to/dir2'
    test_results "$res" "$expected"

    res=$(relpath_ /path1/to/dir2 / )
    expected='../..'
    test_results "$res" "$expected"

    res=$(relpath_ / / )
    expected='.'
    test_results "$res" "$expected"

    res=$(relpath_ /path/to/dir2/dir3 /path/to/dir1/dir4/dir4a )
    expected='../..../dir1/dir4/dir4a'
    test_results "$res" "$expected"

    res=$(relpath_ /path/to/dir1/dir4/dir4a /path/to/dir2/dir3 )
    expected='../..../dir2/dir3'
    test_results "$res" "$expected"

    #res=$(relpath_ . /path/to/dir2/dir3 )
    #expected='../..../dir2/dir3'
    #test_results "$res" "$expected"
}

test_results () {
    if test ! "$1" = "$2"; then
        printf 'failed!\nresult:\nX%sX\nexpected:\nX%sX\n\n' "$@"
    fi
}

#relpath_test

```

ИСТОЧНИК: <http://www.ynform.org/w/Pub/Relpath>

Делиться Улучшить этот ответ
Следовать

отредактировано 8 сентября 2011
г. в 13:05

ответил 3 мая 2010 г. в 12:41



Бедный Йорик
263 1 4 4

- 1 Это не совсем переносимо из-за использования конструкции `${param/pattern/subst}`, которая не соответствует POSIX (по состоянию на 2011 год). – Йенс 11 авг 2011 в 11:57

Указанный источник ynform.org/w/Pub/Relpath указывает на полностью искаженную страницу вики, содержащую содержимое скрипта несколько раз, перемежающееся строками `vi` с тильдами, сообщениями об ошибках о том, что команды не найдены и т. д. Совершенно бесполезно для того, кто исследует оригинал. – Йенс 11 авг 2011 в 15:02



-1



Мне нужно было что-то вроде этого, но разрешающее также символические ссылки. Я обнаружил, что у `pwd` есть флаг `-P` для этой цели. Фрагмент моего скрипта добавлен. Он находится внутри функции в скрипте оболочки, отсюда `$1` и `$2`. Значение результата, которое является относительным путем от `START_ABS` до `END_ABS`, находится в переменной `UPDIRS`. Скрипт переходит в каждый каталог параметров, чтобы выполнить `pwd -P`, и это также означает, что обрабатываются параметры относительного пути. Ура, Джим

```
SAVE_DIR="$PWD"
cd "$1"
START_ABS=`pwd -P`
cd "$SAVE_DIR"
cd "$2"
END_ABS=`pwd -P`

START_WORK="$START_ABS"
UPDIRS=""

while test -n "${START_WORK}" -a "${END_ABS/${START_WORK}} " '==' "$END_ABS";
do
    START_WORK=`dirname "$START_WORK"`"/"
    UPDIRS="${UPDIRS}".. "/"
done
UPDIRS="$UPDIRS${END_ABS/${START_WORK}} "
cd "$SAVE_DIR"
```

Делиться Улучшить этот ответ Следовать

ответил 29 мая 2014 в 12:43



СЖЖХ

51 1 2



Очень активный вопрос . Заработайте 10 очков репутации (не считая бонуса ассоциации), чтобы ответить на этот вопрос. Требование репутации помогает защитить этот вопрос от спама и неответов.