The printf command

Fix Me! Stranger, this is a very big topic that needs experience - please fill in missing information, extend the descriptions, and correct the details if you can!

Attention:

This is about the Bash-builtin command `printf` - however, the description should be nearly identical for an external command that follows POSIX®.

GNU Awk (http://www.gnu.org/software/gawk/manual/gawk.html#Printf) expects a comma after the format string and between each of the arguments of a **printf** command. For examples, see: **code snippet**.

Unlike other documentations, I don't want to redirect you to the manual page for the `printf()` C function family. However, if you're more experienced, that should be the most detailed description for the format strings and modifiers.

Due to conflicting historical implementations of the `echo` command, POSIX® recommends that `printf` is preferred over `echo`.

General

`printf` Команда предоставляет метод для печати предварительно отформатированного текста, аналогичный `printf()` системному интерфейсу (функция C). Она предназначена в качестве преемника `echo` и имеет гораздо больше функций и возможностей.

Помимо других причин, у POSIX® есть очень веский аргумент, чтобы рекомендовать ее: оба исторических основных варианта `echo` команды являются взаимоисключающими, они сталкиваются. Для решения проблемы пришлось изобрести "новую" команду.

Синтаксис

```
printf <ФОРМАТ> <АРГУМЕНТЫ ...>
```

Текстовый формат задается в `<FORMAT>` , в то время как все аргументы, на которые может указывать formatstring, задаются после этого, здесь, обозначается `<ARGUMENTS…>` .

Таким образом, типичный `printf` вызов выглядит следующим образом:

```
printf "Фамилия: %s\nName: %s \n" "$ФАМИЛИЯ" "$ИМЯ ПОЛЬЗОВАТЕЛЯ"
```

где `"Surname: %s\nName: %s\n"` находится спецификация формата, и две переменные передаются в качестве аргументов, `%s` на которые указывает строка формата (для каждого заданного вами спецификатора формата `printf` ожидает один аргумент!).

Опции

`-v VAR`	Если задано, выходные данные присваиваются переменной `VAR` , а не печатаются `stdout` (сопоставимы `sprintf()` с каким-либо образом)

`-v` Опция не может назначаться напрямую индексам массива в версиях Bash старше версии 4.1.

В версиях, более новых, чем 4.1, нужно быть осторожным при выполнении расширений в первый параметр, не являющийся параметром printf, поскольку это открывает возможность легкой уязвимости при внедрении кода.

```
$ var='-vx[$(echo hi >&2)]'; printf "$var" привет; объявить -p x
привет
объявить -a x='([0]="привет")'
```

... где echo, конечно, можно заменить любой произвольной командой. Если необходимо, либо укажите жестко заданную строку формата, либо используйте --, чтобы указать конец параметров. Точно такая же проблема также относится к read и аналогична mapfile , хотя выполнение расширений в их аргументы встречается реже.

Аргументы

Конечно, в оболочке - это означает, что аргументы являются просто строками, однако общие обозначения C плюс некоторые дополнения для числовых констант распознаются для предоставления числового аргумента `printf` :

Числовой формат	**Описание**
`N`	Обычное десятичное число
`0N`	Восьмеричное число
`0xN`	Шестнадцатеричное число

Числовой формат	Описание
`0XN`	Шестнадцатеричное число
`"X`	(буквальная двойная кавычка перед символом): интерпретируется как число (базовый набор кодов) **не забудьте экранировать**
`'X`	(буквальная одинарная кавычка перед символом): интерпретируется как число (базовый набор кодов) **не забудьте экранировать**

Если аргументов больше, чем спецификаторов формата, то строка формата используется повторно, пока не будет интерпретирован последний аргумент. Если спецификаторов формата меньше, чем аргументов, то числовым форматам присваивается значение ноль, а строковым форматам присваивается значение null (пустой).

Старайтесь избегать разделения слов, так как случайная передача неправильного количества аргументов может привести к совершенно разным и неожиданным результатам. См. Эту статью.

Опять же, внимание: когда числовой формат ожидает число, внутренняя `printf` команда будет использовать общие правила арифметики Bash в отношении базы. Команда, подобная следующему примеру, выдаст ошибку, поскольку `08` не является допустимым восьмеричным числом (`00` to `07` !):

```
printf '%d\n' 08
```

Форматировать строки

Интерпретация строки формата является производной от семейства `printf()` функций C. Распознаются только спецификаторы формата, которые заканчиваются на одну из букв `diouxXfeEgGaAcs` .

Чтобы напечатать литерал `%` (знак процента), используйте `%%` в строке формата.

Опять же: каждый спецификатор формата ожидает предоставления соответствующего аргумента!

Эти спецификаторы имеют разные имена, в зависимости от того, кого вы спрашиваете. Но все они означают одно и то же: заполнитель для данных с указанным форматом:

- формат заполнителя
- спецификация преобразования
- форматирование токена
- …

Форматировать	Описание
`%b`	Выведите соответствующий аргумент при интерпретации экранирования обратной косой черты
`%q`	Выведите соответствующий аргумент в **кавычках**, повторно используемый в качестве входных данных

Форматировать	Описание
`%d`	Выведите соответствующий аргумент в виде **десятичного числа со знаком**
`%i`	То же, что `%d`
`%o`	Выведите соответствующий аргумент в виде **восьмеричного числа без знака**
`%u`	Выведите соответствующий аргумент в виде **десятичного** числа без знака
`%x`	Выведите соответствующий аргумент в виде **шестнадцатеричного** числа без знака со строчными шестнадцатеричными цифрами (a-f)
`%X`	То же, `%x` что и, но с шестнадцатеричными цифрами в верхнем регистре (A-F)
`%f`	Интерпретируйте и распечатайте связанный аргумент как число с **плавающей запятой**
`%e`	Интерпретируйте связанный аргумент как **double** и распечатайте его в `<N>±e<N>` формате
`%E`	То же, `%e` что и, но с прописными буквами `E` в печатном формате
`%g`	Интерпретирует связанный аргумент как **double**, но печатает его как `%f` или `%e`
`%G`	То же `%g` , что и, но напечатать его как `%E`
`%c`	Интерпретирует связанный аргумент как **символ**: печатается только первый символ данного аргумента
`%s`	Интерпретирует связанный аргумент буквально как строку
`%n`	Присваивает количество символов, напечатанных на данный момент, переменной, указанной в соответствующем аргументе. Не удается указать индекс массива. Если заданное имя уже является массивом, значение присваивается нулевому элементу.
`%a`	Интерпретирует связанный аргумент как **double** и выводит его в виде шестнадцатеричного литерала C99 с плавающей запятой (http://www.exploringbinary.com/hexadecimal-floating-point-constants/).
`%A`	То же `%a` , что и, но напечатать его как `%E`
`%(FORMAT)T`	выведите строку даты и времени, полученную в результате использования `FORMAT` в качестве строки формата для `strftime(3)` . Связанный аргумент - это количество секунд, прошедших с момента Epoch, или `-1` (текущее время) или `-2` (время запуска оболочки). Если соответствующий аргумент не указан, по умолчанию используется текущее время
`%%`	Преобразование не выполняется. Выдает `%` (знак процента)

Некоторые из упомянутых спецификаторов формата могут изменять свое поведение, получая модификатор формата:

Модификаторы

Для большей гибкости при выводе чисел и строк `printf` команда допускает модификаторы формата. Они указаны **между** вводным `%` и символом, который определяет формат:

```
printf "%50s\n" "Это поле имеет ширину 50 символов ..."
```

Модификаторы полей и печати

Формат вывода поля	
`<N>`	**Любое число**: задает **минимальную ширину поля**, если текст для печати короче, он дополняется пробелами, если текст длиннее, поле расширяется
`.`	**Точка**: вместе с шириной поля поле **не** расширяется, когда текст длиннее, вместо этого текст усекается. "`%.s`" - это недокументированный эквивалент "`%.0s`", который принудительно принимает ширину поля равной нулю, эффективно скрывая поле от вывода
`*`	**Звездочка**: ширина указывается в качестве аргумента перед строкой или числом. Использование ("`*`" соответствует "`20`"): `printf "%*s\n" 20 "test string"`
`#`	"Альтернативный формат" для чисел: см. Таблицу ниже
`-`	**Печать текста в поле с привязкой к левому** краю (стандартная **привязка к правому** краю)
`0`	Дополняет числа нулями, а не пробелами
`<space>`	Дополните положительное число пробелом, где минус (`-`) для отрицательных чисел
`+`	Печатает все числа со **знаком** (`+` для положительных, `-` для отрицательных)
`'`	Для десятичных преобразований разделитель группирования тысяч применяется к целочисленной части выходных данных в соответствии с текущим LC_NUMERIC

Модификатор "Альтернативный формат" `#` :

Альтернативный формат	
`%#o`	Восьмеричное число печатается с начальным нулем, если только оно само не равно нулю

Альтернативный формат	
`%#x , %#X`	Шестнадцатеричное число печатается с начальным " `0x` "/" `0X` ", если оно не равно нулю
`%#g , %#G`	Число с плавающей запятой печатается с **конечными нулями** до тех пор, пока не будет достигнуто количество цифр для текущей точности (обычно конечные нули не печатаются)
все числовые форматы, кроме `%d , %o , %x , %X`	Всегда выводите десятичную точку в выходных данных, даже если за ней не следует никаких цифр

Точность

Точность для плавающего или двойного числа может быть указана с помощью `.<DIGITS>` , где `<DIGITS>` - количество цифр для точности. Если `<DIGITS>` это звездочка (`*`), точность считывается из аргумента, который предшествует числу для печати, например (печатает 4,3000000000):

```
printf "%.*f \n" 10 4,3
```

Формат `.*N` для указания N-го аргумента для точности не работает в Bash.

Для строк точность определяет максимальное количество символов для печати (т. Е. Максимальную ширину поля). Для целых чисел она указывает количество цифр для печати (заполнение нулем!).

Escape-коды

Они интерпретируются, если используются в любом месте строки формата или в аргументе, соответствующем `%b` формату.

Код	**Описание**
`\\`	Печатает символ `\` (обратную косую черту)
`\a`	Печатает символ предупреждения (код ASCII () 7 десятичный)
`\b`	Печатает пробел
`\f`	Печатает ленту формы
`\n`	Выводит новую строку
`\r`	Печатает возврат каретки
`\t`	Печатает горизонтальный табулятор
`\v`	Печатает вертикальный табулятор
`\"`	Печатает `'`
`\?`	Печатает `?`

Код	Описание
`\<NNN>`	Интерпретирует `<NNN>` как **восьмеричное** число и печатает соответствующий символ из набора символов
`\0<NNN>`	то же, что `\<NNN>`
`\x<NNN>`	Интерпретирует `<NNN>` как **шестнадцатеричное** число и печатает соответствующий символ из набора символов (**3 цифры**)
`\u<NNNN>`	то же `\x<NNN>` , что и, но **4 цифры**
`\U<NNNNNNNN>`	то же `\x<NNN>` , что и, но **8 цифр**

Следующие дополнительные escape и дополнительные правила применяются только к аргументам, связанным с `%b` форматом:

`\c` Завершает вывод аналогично `\c` escape, используемому `echo -e` . printf не выдает дополнительных выходных данных после обнаружения `\c` escape в `%b` аргументе.

- Обратная косая черта в escapes: `\'` , `\"` , и `\?` не удаляется.
- Восьмеричные символы, начинающиеся с `\0` , могут содержать до четырех цифр. (В POSIX указывается до трех).

Это также аспекты, которые `%b` отличаются от экранирования, используемого при цитировании в стиле $'...'.

Примеры

Фрагменты

- выведите десятичное представление шестнадцатеричного числа (сохраните знак)
  - `printf "%d\n" 0x41`
  - `printf "%d\n" -0x41`
  - `printf "%+d\n" 0x41`
- выведите восьмеричное представление десятичного числа
  - `printf "%o\n" 65`
  - `printf "%05o\n" 65` (Ширина 5 символов, дополненных нулями)
- при этом выводится значение 0, поскольку аргумент не указан
  - `printf "%d\n"`
- выведите кодовый номер символа `A`
  - `printf "%d\n" \'A`
  - `printf "%d\n" "'A"`
- Создайте приветствующий баннер и присвоите его переменной `GREETER`
  - `printf -v GREETER "Hello %s" "$LOGNAME"`
- Выведите текст в конце строки, используя `tput` для получения текущей ширины строки

- `printf "%*s\n" $(tput cols) "Hello world!"`

Небольшая таблица кода

Этот небольшой цикл печатает все числа от 0 до 127 в

- десятичная
- Восьмеричный
- Шестнадцатеричный

```
для ((x=0; x <= 127; x++)); выполнить
printf '%3d | %04o | 0x%02x \n' "$x" "$x" "$x"
готово
```

Убедитесь, что MAC-адрес правильно отформатирован

Этот код здесь возьмет обычный MAC-адрес и перепишет его в хорошо известный формат (в отношении начальных нулей или верхнего / нижнего регистра шестнадцатеричных цифр, ...):

```
the_mac="0:13:ce:7:7a:ad"

# шестнадцатеричные цифры
в нижнем регистре the_mac="$(printf "%02x:%02x:%02x:%02x:%02x" 0x${th
e_mac//:/ 0x})"

# или вариант
с прописными цифрами the_mac="$(printf "%02X:%02X:%02X:%02X:%02X:%02
X" 0x${the_mac//:/ 0x})"
```

Замена echo

Этот код был найден на странице руководства Solaris для echo(1).

Версия Solaris `/usr/bin/echo` эквивалентна:

```
printf "%b\n" "$*"
```

Solaris `/usr/ucb/echo` эквивалентен:

```
если [ "X $ 1" = "X-n" ]
, то
сдвиньте
printf "%s" "$*"
иначе
printf "%s\n" "$*"
fi
```

Реализация prargs

Отрабатывая эхо-сигнал замены, вот краткая реализация prargs:

```
printf '"%b"\n' "$0" "$@" | nl -v0 -s": "
```

повторение символа (например, для печати строки)

Небольшая хитрость: объединение printf и расширения параметров для рисования линии

```
длина =40
строка printf -v '%*s' "$length"
echo ${строка// /-}
```

или:

```
длина =40
вычислить строку printf -v '%.0s-' {1..$length}
```

Замена некоторых вызовов на сегодняшний день(1)

Строка `%(…)T` формата является прямым интерфейсом к `strftime(3)`.

```
$ printf 'Это неделя %(%U/%Y)T.\n' -1
Это неделя 52/2010.
```

Пожалуйста, прочтите справочную `strftime(3)` страницу, чтобы получить больше информации о поддерживаемых форматах.

отличия от awk printf

Awk также получает свою функцию *printf()* из C и, следовательно, имеет аналогичные спецификаторы формата. Однако во всех версиях awk символ пробела используется в качестве оператора конкатенации строк, поэтому его нельзя использовать в качестве разделителя аргументов. **Аргументы в awk printf должны разделяться запятыми.**Некоторые версии awk не требуют, чтобы аргументы printf заключались в круглые скобки, но вы должны использовать их в любом случае для обеспечения переносимости.

В следующем примере две строки объединяются промежуточным пробелом, чтобы не осталось аргументов для заполнения формата.

```
$ echo "Foo" | awk '{ printf "%s \n" $1 }'
awk: (FILENAME=- FNR=1) фатальный: недостаточно аргументов для соотве
тствия строке формата
 `%s
Foo'
 ^ закончилась для этого
```

Простая замена пробела запятой и добавление круглых скобок дает правильный синтаксис awk.

```
$ echo "Foo" | awk '{ printf( "%s \n", $1 ) }'
Foo
```

При соответствующем экранировании метасимвола bash printf может быть вызван изнутри awk (как из perl, так и из других языков, поддерживающих выноску оболочки), если вас не волнует эффективность или удобочитаемость программы.

```
echo "Foo" | awk '{ system( "printf"%s \\n \" \"" $1 "\"" ) }'
Фу
```

Отличия от C и соображения переносимости

- Преобразования a, A, e, E, f, F, g и G поддерживаются Bash, но не требуются POSIX.
- Нет поддержки широкосимвольных символов (wprintf). Например, если вы используете `%c`, вы фактически запрашиваете первый байт аргумента. Аналогично, модификатор максимальной ширины поля (точка) в сочетании с `%s` указывается в байтах, а не в символах. Это ограничивает некоторые функциональные возможности printf только для работы с ascii. ksh93 `printf` поддерживает `L` модификатор с `%s` and `%c` (но пока не `%S` or `%C`), чтобы обрабатывать точность как ширину символа, а не количество байтов. похоже, что zsh динамически настраивается на основе `LANG` и `LC_CTYPE`. Если `LC_CTYPE=C`, zsh выдаст ошибку "символ не в диапазоне", и в противном случае автоматически поддерживает широкие символы, если для текущей локали установлена кодировка переменной ширины.
- Bash распознает и пропускает любые символы, присутствующие в модификаторах длины, указанных POSIX, во время синтаксического анализа строки формата.

встроенные /printf.def

```
#определение LENMODS "hjlLtz"
...
/* пропустить возможные модификаторы формата */
modstart = fmt;
в то время  как(*fmt && strchr (LENMODS, *fmt))
fmt++;
```

- у mksh нет встроенного printf по умолчанию (обычно). Существует неподдерживаемая опция времени компиляции, включающая очень плохую, в основном непригодную для использования реализацию. По большей части вы должны полагаться на систему `/usr/bin/printf` или ее эквивалент. Сопровождающий mksh рекомендует использовать `print` . Версия разработки (после R40f) добавляет новый параметр expansion, форма `${name@Q}` которого выполняет роль `printf %q` – expanding в формате, экранируемом оболочкой.
- ksh93 оптимизирует встроенные функции, запускаемые из командной подстановки и не имеющие перенаправлений для запуска в процессе оболочки. Поэтому `printf -v` функциональность может быть точно подобрана `var=$(printf …)` без большого снижения производительности.

```
# Иллюстрирует поведение, подобное Bash. Переопределение printf обычн
о не требуется / не рекомендуется.
функция printf {
 случай $ 1 в
 -v)
shift
nameref x=$1
 сдвиг
 x=$(команда printf "$@")
 ;;
 *)
команда printf "$@"
 esac
}
встроенный cut
print $$
printf -v 'foo[2]' '%d \n' "$(cut -d ' ' -f 1 /proc/self/stat)"
набор текста -p foo
# 22461
# typeset -a foo=([2]=22461)
```

- Дополнительный загрузчик Bash `print` может быть полезен для совместимости с ksh и преодоления некоторых недостатков переносимости echo. У Bash, ksh93 и zsh `print` есть `-f` опция, которая принимает строку `printf` формата и применяет ее к оставшимся аргументам. Bash перечисляет краткий обзор следующим образом: `print: print [-Rnprs] [-u unit] [-f format] [arguments]` . Однако `-Rrnfu` на самом деле функционируют только. Внутренне `-p` это noop (он вообще не связан с Bash coprocs) и `-s` только устанавливает флаг, но не имеет никакого эффекта. `-Cev` не выполняется.
- Присвоение переменным: `printf -v` способ немного отличается от способа использования подстановки команд. Замена команды удаляет завершающие новые строки перед заменой текста, `printf -v` сохраняет весь вывод.

Смотрите также

- SUS: утилита printf (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/printf.html) и функция

printf() (http://pubs.opengroup.org/onlinepubs/9699919799/functions/printf.html)
- Фрагмент кода: печать горизонтальной линии использует несколько `printf` примеров
- BashFAQ Грега 18: Как я могу использовать числа с начальными нулями в цикле, например, 01, 02? (http://mywiki.wooledge.org/BashFAQ/018)

Обсуждение

Дэн Дуглас, 2011/09/10 03:36 ()

Несколько вопиющим упущением в каждой оболочке, которую я должен протестировать, кроме ksh93, являются спецификаторы преобразования нумерованных аргументов. %n$ или *n$ http://pubs.opengroup.org/onlinepubs/9699919799/functions/printf.html (http://pubs.opengroup.org/onlinepubs/9699919799/functions/printf.html)

Идея состоит в том, чтобы разрешить либо изменение порядка аргументов, либо повторное использование модификатора ширины (с *), путем определения того, какие преобразования применяются к каким аргументам. К сожалению, это на самом деле не избавляет вас от необходимости вводить текст, потому что нумерация любых преобразований приводит к тому, что поведение, когда аргументов больше, чем преобразований, исчезает. В спецификации указано, что поведение в этом случае не определено. Ksh просто выполняет сегментацию (как в 3-м примере ниже). Первый и второй, по сути, эквивалентны.

```
~ $ printf '%.*s\ n' 3 'foobar' 3 'foobarbaz' #Bash
foo
foo
~ $ ksh
$ printf '%.* 1 $ s \ n%3 $.* 1 $ s \ n ' 3 'foobar' 'foobarbaz'
foo
foo
$ printf '%.* 1$ s \ n' 3 'foobar' 'foobarbaz'
Ошибка сегментации
~ $ printf '%.* 1 $ s \ n%3 $.*1 $ s \ n' 3 'foobar' 'foobarbaz'
#Вернуться в Bash
-bash: printf: `1': недопустимый символ формата
~ $
```

Довольно глупо, как это указано, но я могу представить несколько сценариев, в которых это может быть полезно. И учитывая, что Bash просто указывает на printf(3) в качестве документации, это довольно странная вещь, которую можно пропустить, хотя она также отсутствует почти во всех других встроенных оболочках плюс printf(1) gnu coreutils. (Это также описано в printf(3) на страницах руководства по Linux, в котором упоминается, что это SUS, но не C99.)

Альтаир IV, 05.02.2012 14:19 ()

Ммм, что? Этому последнему разделу здесь нечего делать. встроенная функция printf awk полностью отличается от версии оболочки. Синтаксис для printf awk разделен запятыми, то есть 'printf("<format>" , "<arguments>")' , при этом круглые скобки являются необязательными. Так что на самом деле это первое "исправление", которое использует его правильно.

Дэн Дуглас, 07.07.2012 / 12 06:13 ()

Так что исправьте это, это вики. :)

MJF, 07.07.2012 12:18 ()

> Этому последнему разделу здесь нечего делать. встроенная функция printf awk полностью отличается от версии оболочки.

Да, чертовски очевидно, что "printf" и "printf" - это "совершенно разные" вещи. Тот факт, что синтаксис для них лишь немного отличается, является просто массовым совпадением и не может привести к какой-либо путанице. И зачем беспокоиться о документировании этой разницы? Люди должны просто прочитать исходный код, чтобы понять, почему awk выдает ошибки, верно?

Р.У. Эмерсон II, 2012/12/08 18:07 ()

Я обнаружил, что следующие строки дают разные результаты:

```
var=$(printf ...)
printf -v переменная ...
```

Например, в первой строке ниже опускается завершающий символ \n, но во второй строке он сохраняется:

```
объявляем vT ; vT=$(printf "%s \n" "ABC") ; echo "vT($ vT)"
объявляем vT ; printf -v vT "%s \n" "ABC" ; echo "vT($ vT)"
```

Приятно, наконец, найти сайт, который подробно документирует эти команды bash. Я часами тщетно искал статью или сообщение, в которых упоминается вышеупомянутая ошибка / функция printf. Я удивлен, обнаружив, что никто больше не упоминал об этом.

Ян Шампера, 2012/12/16 13:19 ()

Причина в том, что подстановка команды `$()` сокращает завершающую новую строку, как упоминалось в статье о подстановке команд.

Таким образом, ваше уведомление абсолютно верно. Эти две команды дают несколько разные результаты, и я должен упомянуть об этом выше.

commands/builtin/printf.txt Последнее изменение: 2016/11/30 15:39 автор medievalist