Массивы

Цель

Массив - это параметр, который содержит сопоставления ключей со значениями. Массивы используются для хранения набора параметров в параметре. Массивы (на любом языке программирования) являются полезной и распространенной составной структурой данных и одной из наиболее важных скриптовых функций в Bash и других оболочках.

Вот **абстрактное** представление массива с именем `NAMES`. Индексы идут от 0 до 3.

```
ИМЕНА
 0: Питер
 1: Анна
 2: Грег
 3 января
```

Вместо использования 4 отдельных переменных, несколько связанных переменных группируются в *элементы* массива, доступные по их *ключу*. Если вам нужно второе имя, запросите индекс 1 массива `NAMES`.

Индексирование

Bash поддерживает два разных типа одномерных массивов, подобных ksh. **Многомерные массивы не реализованы**.

- *Индексированные массивы* используют положительные целые числа в качестве ключей. Индексированные массивы **всегда разрежены**, что означает, что индексы не обязательно являются смежными. Весь синтаксис, используемый как для назначения, так и для разыменования индексированных массивов, представляет собой контекст арифметической оценки (см. Ссылки). Как и в C и многих других языках, числовые индексы массива начинаются с 0 (нуля). Индексированные массивы являются наиболее распространенным, полезным и переносимым типом. Индексированные массивы впервые были представлены в оболочках, подобных Bourne, ksh88. Подобный, частично совместимый синтаксис был унаследован многими производными, включая Bash. Индексированные массивы всегда несут `-a` атрибут.
- *Ассоциативные массивы* (иногда известные как "хэш" или "dict") используют произвольные непустые строки в качестве ключей. Другими словами, ассоциативные массивы позволяют вам искать значение из таблицы на основе соответствующей строковой метки. **Ассоциативные массивы всегда**

неупорядочены, они просто *связывают* пары ключ-значение. Если вы извлекаете несколько значений из массива одновременно, вы не можете рассчитывать на то, что они будут выводиться в том же порядке, в котором вы их ввели. Ассоциативные массивы всегда содержат `-A` атрибут, и в отличие от индексированных массивов, Bash требует, чтобы они всегда объявлялись явно (поскольку по умолчанию используются индексированные массивы, см. Объявление). Ассоциативные массивы были впервые представлены в ksh93, и аналогичные механизмы были позже приняты Zsh и Bash версии 4. В настоящее время эти три являются единственными POSIX-совместимыми оболочками с любой поддержкой ассоциативных массивов.

Синтаксис

Ссылки

Чтобы приспособить ссылки на переменные массива и их отдельные элементы, Bash расширяет схему именования параметров с помощью суффикса нижнего индекса. Любое допустимое имя обычного скалярного параметра также является допустимым именем массива : `[[:alpha:]_][[:alnum:]_]*` . За именем параметра может следовать необязательный нижний индекс, заключенный в квадратные скобки для обозначения элемента массива.

Общий синтаксис `arrname[subscript]` - где для индексированных массивов - `subscript` любое допустимое арифметическое выражение, а для ассоциативных массивов - любая непустая строка. Индексы сначала обрабатываются для расширения параметров и арифметических операций, а также для подстановок команд и процессов. При использовании в расширениях параметров или в качестве аргумента для встроенного параметра unset также принимаются специальные индексы `*` и `@` , которые воздействуют на массивы аналогично тому `@` , `*` как специальные параметры и воздействуют на позиционные параметры. При синтаксическом анализе нижнего индекса bash игнорирует любой текст, который следует за закрывающей скобкой до конца имени параметра.

За редким исключением, имена этой формы могут использоваться везде, где допустимы обычные имена параметров, например, в арифметических выражениях, расширениях параметров и в качестве аргументов встроенных функций, которые принимают имена параметров. *Массив* - это параметр Bash, которому заданы `-a` (для индексированных) или `-A` (для ассоциативных) *атрибутов*. Однако на любой обычный (не специальный или позиционный) параметр можно корректно ссылаться с использованием нижнего индекса, поскольку в большинстве контекстов ссылка на нулевой элемент массива является синонимом ссылки на имя массива без нижнего индекса.

```
# "x" - это обычный параметр, не относящийся к массиву.
$ x=hi; printf '%s ' "$x" "${x[0]}"; echo "${_[0]}"
привет, привет, привет
```

Единственными исключениями из этого правила являются несколько случаев, когда имя переменной массива ссылается на массив в целом. Это относится к `unset` встроенному (см. Уничтожение) и при объявлении массива без присвоения каких-либо значений (см. Объявление).

Объявление

Следующие переменные явно задают атрибуты массива, делая их массивами:

Синтаксис	Описание
`ARRAY=()`	Объявляет **индексированный** массив `ARRAY` и инициализирует его как пустой. Это также можно использовать для очистки существующего массива.
`ARRAY[0]=`	Обычно задает первый элемент **индексированного** массива. Если массив не `ARRAY` существовал ранее, он создается.
`declare -a ARRAY`	Объявляет **индексированный** массив `ARRAY`. Существующий массив не инициализируется.
`declare -A ARRAY`	Объявляет **ассоциативный** массив `ARRAY`. Это единственный и неповторимый способ создания ассоциативных массивов.

В качестве примера и для использования ниже давайте объявим наш `NAMES` массив, как описано выше:

```
объявить ИМЕНА =('Питер' 'Анна' 'Грег' 'Ян')
```

Хранение значений

Хранить значения в массивах так же просто, как хранить значения в обычных переменных.

Синтаксис	Описание
`ARRAY[N]=VALUE`	Задает элементу `N` **индексированного** массива `ARRAY` значение `VALUE`. **`N` может быть любым допустимым арифметическим выражением**.
`ARRAY[STRING]=VALUE`	Задает элемент, индексируемый `STRING` **ассоциативным массивом** `ARRAY`.
`ARRAY=VALUE`	Как указано выше. Если индекс не указан, по умолчанию нулевому элементу присваивается значение `VALUE`. Осторожно, это справедливо даже для ассоциативных массивов - ошибки не будет, если ключ не указан, а значение присваивается строковому индексу "0".

Синтаксис	Описание
`ARRAY=(E1 E2 …)`	Назначение составного массива - присваивает всему массиву `ARRAY` заданный список элементов, индексируемых последовательно, начиная с нуля. Массив не устанавливается перед присваиванием, если не используется оператор += . Когда список пуст (`ARRAY=()`), массив будет установлен в пустой массив. Очевидно, что этот метод не использует явные индексы. **Ассоциативный массив не** может быть установлен таким образом! Очистка ассоциативного массива с помощью `ARRAY=()` works.
`ARRAY=([X]=E1 [Y]=E2 …)`	Составное присваивание для индексированных массивов с парами индекс-значение, объявленными индивидуально (здесь, например `X` , и `Y`). X и Y являются арифметическими выражениями. Этот синтаксис можно комбинировать с приведенным выше - элементы, объявленные без явно указанного индекса, присваиваются последовательно, начиная либо с последнего элемента с явным индексом, либо с нуля.
`ARRAY=([S1]=E1 [S2]=E2 …)`	Индивидуальная настройка массы для **ассоциативных массивов**. Именованные индексы (здесь: `S1` и `S2`) являются строками.
`ARRAY+=(E1 E2 …)`	Добавить в МАССИВ.
`ARRAY=("${ANOTHER_ARRAY[@]}")`	Скопируйте ANOTHER_ARRAY в МАССИВ, копируя каждый элемент.

На данный момент массивы нельзя экспортировать.

Получение значений

Для полноты и подробностей о нескольких вариантах расширения параметров см. Статью о расширении параметров и проверьте Примечания о массивах.

Синтаксис	Описание
`${ARRAY[N]}`	Расширяется до значения индекса `N` в **индексированном** массиве `ARRAY` . Если `N` это отрицательное число, оно обрабатывается как смещение от максимального присвоенного индекса (не может использоваться для присвоения) - 1
`${ARRAY[S]}`	Расширяется до значения индекса `S` в **ассоциативном** массиве `ARRAY` .

Синтаксис	Описание

`"${ARRAY[@]}"`
`${ARRAY[@]}`
`"${ARRAY[*]}"`
`${ARRAY[*]}`
	Подобно позиционным параметрам с массовым расширением, это распространяется на все элементы. Если кавычки не указаны, оба индекса `*` и `@` расширяются до одного и того же результата, если `@` они заключены в кавычки, расширяются до всех элементов, указанных в отдельности `*`, расширяются до всех элементов, указанных в целом.

`"${ARRAY[@]:N:M}"`
`${ARRAY[@]:N:M}`
`"${ARRAY[*]:N:M}"`
`${ARRAY[*]:N:M}`
	Подобно тому, что этот синтаксис делает для символов одной строки при расширении подстроки, он расширяется до `M` элементов, начинающихся с element `N`. Таким образом, вы можете массово расширять отдельные индексы. Правила для цитирования и индексов `*` и `@` такие же, как указано выше для других массовых расширений.

Для пояснения: когда вы используете индексы `@` или `*` для массового расширения, тогда поведение является именно тем, для чего оно предназначено `$@`, и `$*` при массовом расширении позиционных параметров. Вы должны прочитать эту статью, чтобы понять, что происходит.

Метаданные

Синтаксис	Описание
`${#ARRAY[N]}`	Расширяется до **длины** отдельного элемента массива по индексу `N` (**stringlength**)
`${#ARRAY[STRING]}`	Расширяется до **длины** отдельного ассоциативного элемента массива по индексу `STRING` (**stringlength**)
`${#ARRAY[@]}` `${#ARRAY[*]}`	Расширяется до **количества элементов** `в ARRAY`
`${!ARRAY[@]}` `${!ARRAY[*]}`	Расширяется до **индексов** `ARRAY`, начиная с BASH 3.0

Разрушение

Встроенная команда unset используется для уничтожения (отмены установки) массивов или отдельных элементов массивов.

Синтаксис	Описание

`unset -v ARRAY`
`unset -v ARRAY[@]`
`unset -v ARRAY[*]`
	Уничтожает полный массив

`unset -v ARRAY[N]`	Уничтожает элемент массива по индексу `N`

Синтаксис	Описание
`unset -v ARRAY[STRING]`	Уничтожает элемент массива ассоциативного массива по индексу `STRING`

Лучше всего явно указывать -v при сбросе переменных с помощью unset .

Указание элементов массива без кавычек в качестве аргументов для любой команды, например, с помощью синтаксиса, приведенного выше, **может привести к расширению имени пути из-за присутствия символов glob**.
Пример: вы находитесь в каталоге с именем файла `x1` и хотите уничтожить элемент массива `x[1]` с помощью

```
сбросить значение x[1]
```

тогда расширение имени пути расширится до имени `x1` файла и нарушит вашу обработку!

Что еще хуже, если `nullglob` установлено значение, ваш массив / индекс исчезнет.

Чтобы избежать этого, **всегда указывайте имя и индекс массива в кавычках**:

```
сбросить значение -v 'x[1]'
```

Обычно это относится ко всем командам, которые принимают имена переменных в качестве аргументов. Предпочтительны одинарные кавычки.

Использование

Числовой индекс

Числовые индексированные массивы просты для понимания и просты в использовании. Приведенные выше главы о назначении и индексации более или менее объясняют всю необходимую базовую теорию.

Теперь несколько примеров и комментариев для вас.

Допустим, у нас есть массив `sentence` , который инициализируется следующим образом:

```
предложение = (Будьте либеральны в том, что вы принимаете, и консерва
тивны в том, что вы отправляете)
```

Поскольку не существует специального кода для предотвращения разделения слов (без кавычек), каждое слово там будет присвоено отдельному элементу массива. Когда вы считаете слова, которые видите, у вас должно получиться 12. Теперь давайте посмотрим, придерживается ли Bash того же мнения:

```
$ echo ${#предложение[@]}
12
```

Да, 12. Хорошо. Вы можете использовать это число для обхода массива. Просто **вычтите 1 из числа элементов и начните свой путь с 0 (нуля)**:

```
((n_elements=${#предложение[@]}, max_index=n_elements - 1))

для ((i = 0; i <= max_index; i++)); выполнить
эхо "Элемент $i: '${предложение [i]}'"
выполнено
```

Вы всегда должны помнить, что, похоже, у новичков иногда возникают проблемы. Пожалуйста, поймите, что **числовая индексация массива начинается с 0 (нуля)**!

Описанный выше метод, проходящий через массив, просто зная его количество элементов, работает только для массивов, где все элементы заданы, конечно. Если удалить один элемент в середине, то вычисление бессмысленно, потому что количество элементов больше не соответствует наивысшему используемому индексу (мы называем их "*разреженными массивами*").

Теперь предположим, что вы хотите заменить свой массив `sentence` значениями из ранее объявленного массива `NAMES` . Вы могли бы подумать, что могли бы просто сделать

```
$ отменить предложение; объявить -предложение = ИМЕНА
$ echo ${#предложение[@]}
1
# опустите вычисление max_index, как указано выше, и выполните итерац
ию как однострочную
$ for ((i = 0; i < ${#предложение[@]}; i++)); выполнить эхо "Элемент
$ i: '${предложение [i]}'" ; готово
Элемент 0: 'ИМЕНА'
```

Очевидно, что это неправильно. Как насчет

```
$ unset предложение ; объявить -предложение=${ИМЕНА}
```

? Опять неправильно:

```
$ echo ${#предложение[*]}
1
$ for ((i = 0; i < ${#предложение[@]}; i++)); выполнить эхо "Элемент
$ i: '${предложение [i]}'" ; готово
Элемент 0: "Питер"
```

Итак, каков **правильный** путь? (Немного уродливый) ответ заключается в повторном использовании синтаксиса перечисления:

```
$ unset предложение ; объявить -предложение=("${ИМЕНА[@]}")
$ echo ${#предложение[@]}
4
$ for ((i = 0; i < ${#предложение[@]}; i++)); выполнить эхо "Элемент
$ i: '${предложение [i]}'" ; готово
Элемент 0: "Питер"
Элемент 1: "Анна"
Элемент 2: 'Грег'
Элемент 3: 'Jan'
```

Ассоциативный (Bash 4)

Ассоциативные массивы (или *хэш-таблицы*) не намного сложнее числовых индексированных массивов. Числовое значение индекса (в Bash число, начинающееся с нуля) просто заменяется произвольной строкой:

```
# declare -A, введенный в Bash 4 для объявления ассоциативного массив
a
declare -предложение

предложение [Начало] = 'Будьте либеральны в том, что'
предложение [Середина]='вы принимаете, и консервативны'
предложение [Конец]= 'в том, что вы отправляете'
предложение ['Самый конец'] =...
```

Осторожно: не полагайтесь на тот факт, что элементы упорядочены в памяти так, как они были объявлены, это может выглядеть так:

```
# вывод из команды 'set'
предложение = ([End]= "в том, что вы отправляете" [Middle] ="вы прини
маете и консервативны " [Begin] ="Будьте либеральны в том, что " ["Са
мый конец"] ="...")
```

Это фактически означает, что вы `"${sentence[@]}"` , конечно, можете получить данные обратно (как и при числовой индексации), но вы не можете полагаться на определенный порядок. Если вы хотите сохранить упорядоченные данные или изменить порядок данных, используйте числовые индексы. Для ассоциативных массивов обычно запрашиваются известные значения индекса:

```
для элемента в начале среднего конца "Самый конец"; сделайте
printf "%s" "${предложение [$элемент]}"
готово
printf "\n"
```

Хороший пример кода: проверка на наличие дубликатов файлов с использованием ассоциативного массива, индексированного суммой файлов SHA:

```
# Спасибо Tramp в #bash за идею и код

сбросить flist; объявить -flist;
при чтении -r sum fname; выполнить
if [[ ${flist[$sum]} ]]; затем
printf 'rm -- "%s" # То же, что >%s<\n' "$fname" "${flist[$sum]}"
 else
flist[$sum]="$fname"
 фи
сделано < <(найти . -введите f -exec sha256sum {} +) > rmdups
```

Целочисленные массивы

Любые атрибуты типа, применяемые к массиву, применяются ко всем элементам массива. Если атрибут integer установлен либо для индексированных, либо для ассоциативных массивов, то значения рассматриваются как арифметические как для составного, так и для обычного присваивания, а оператор += модифицируется таким же образом, как и для обычных целочисленных переменных.

```
 ~ $ ( объявить -ia 'a=(2+4 [2]=2+2 [ a[2]]="a[2]")' 'a+=(42 [a[4]]+=
3)'; объявить -p a )
объявить -ai a='([0]="6" [2]="4" [4]="7" [5]="42")'
```

`a[0]` присваивается результату `2+4` . `a[2]` получает результат `2+2` . Последний индекс в первом присваивании является результатом `a[2]` , который уже был присвоен как `4` , и его значение также задается `a[2]` .

Это показывает, что, хотя любые существующие массивы, названные `a` в текущей области, уже были сброшены с помощью `= +=` составного присваивания вместо, арифметические переменные в ключах могут самостоятельно ссылаться на любые элементы, уже назначенные в том же составном присваивании. С целочисленными массивами это также относится к выражениям справа от `=` . (См. Порядок вычисления, правая часть арифметического задания обычно вычисляется первой в Bash.)

Второй составной аргумент присваивания для объявления использует `+=` , поэтому он добавляется после последнего элемента существующего массива, а не удаляет его и создает новый массив, поэтому `a[5]` получает `42` .

Наконец, элемент, индекс которого является значением `a[4]` (`4`) , `3` добавляется к его существующему значению, создавая `a[4] == 7` . Обратите внимание, что при установке атрибута integer на этот раз += добавляет, а не добавляет строку, как это было бы для нецелочисленного массива.

Одинарные кавычки заставляют задания оцениваться в среде `declare` . Это важно, потому что атрибуты применяются к присваиванию только после обработки аргументов присваивания. Без них `+=` составное присваивание было бы недействительным, и строки были бы вставлены в целочисленный массив без вычисления арифметики. Частный случай этого показан в следующем разделе.

Команды объявления Bash на самом деле являются замаскированными ключевыми словами. Они волшебным образом анализируют аргументы, чтобы определить,

находятся ли они в форме допустимого присваивания. Если это так, они оцениваются как присвоения. Если нет, они подвергаются обычному расширению аргументов перед передачей встроенному, который оценивает результирующую строку как присваивание (что-то вроде `eval` , но есть различия.) `'Todo:` "Обсудите это подробно.

Косвенность

Массивы можно расширять косвенно, используя синтаксис расширения косвенных параметров. Параметры, значения которых имеют вид: `name[index]` , `name[@]` , или `name[*]` при расширении косвенно дают ожидаемые результаты. Это в основном полезно для передачи массивов (особенно нескольких массивов) по имени в функцию.

Этот пример представляет собой предикат, подобный "isSubset", который возвращает true, если все пары ключ-значение массива, заданные в качестве первого аргумента isSubset, соответствуют ключу-значению массива, заданному в качестве второго аргумента. Он демонстрирует как косвенное расширение массива, так и косвенную передачу ключей без eval с использованием вышеупомянутого специального расширения compound assignment .

```
isSubset() {
local -a 'xkeys=("${!'"$1"'[@]}")' ' ключи=("${!'"$2"'[@]}")'
 установить -- "${@/%/[ключ]}"

 (( ${#xkeys[@]} <= ${#ykeys[@]} )) || возвращает 1

 локальный ключ
для ключа в "$ {xkeys[@]}"; сделать
 [[ ${!2+_} && ${!1} == ${!2} ]] || возврат 1
сделано
}

main() {
 # "a" является подмножеством "b"
local -a 'a=({0..5})' ' b=({0..10})'
 isSubset a b
echo $? # верно

 # "a" содержит ключ, которого нет в "b"
локальном - a 'a=([5]=5 {6..11})' ' b=({0..10})'
 isSubset a b
echo $? # ложь

 # "a" содержит элемент, значение которого != соответствующий член
"b"
local -a 'a=([5]=5 6 8 9 10)' ' b=({0..10})'
 isSubset a b
echo $? # ложь
}

Главная
```

Этот скрипт представляет собой один из способов реализации грубого многомерного ассоциативного массива путем хранения определений массива в массиве и ссылки на них через косвенное обращение. Скрипт принимает два ключа и динамически вызывает функцию, имя которой разрешается из массива.

```
callFuncs() {
 # Настройте косвенные ссылки в качестве позиционных параметров, чтоб
ы минимизировать локальные конфликты имен.
 набор -- "${@:1:3}" ${2+' a["$1"]' "$1"'["$2"]'}

 # К сожалению, единственный способ проверить установленные, но нулев
ые параметры - это проверить каждый из них по отдельности.
 локальный x
для x; делать
 [[ $x ]] || возвращает 0
сделано

 локальный -A a=(
[foo]='([r]=f [s]=g [t]=h)'
 [bar]='([u]=i [v]=j [w]=k)'
 [baz]='([x]=l [y]=m [z]=n)'
 ) $ {4+ $ {a["$1"]+"${1}=${!3}"}} # Например, если "$ 1" - это "ba
r", то определите новый массив: bar=([u]=i [v]=j [w]=k)

 ${4+$ {a["$1"]+"${!4-:}"}} # Теперь просто найдите новый массив. для
входных данных: "bar" "v" будет вызвана функция с именем "j", которая
выводит "j" в стандартный вывод.
}

main() {
 # Определите функции с именем {f..n}, которые просто печатают свои с
обственные имена.
 local fun='() { echo "$FUNCNAME"; }' x

 для x в {f..n}; выполните
вычисление "${x}$ {fun}"
 Выполнено

 Функции вызова "$@"
}

главная "$@"
```

Проблемы с ошибками и переносимостью

- Массивы не задаются POSIX. Одномерные индексированные массивы поддерживаются с использованием аналогичного синтаксиса и семантики большинством Korn-подобных оболочек.
- Ассоциативные массивы поддерживаются `typeset -A` в Bash 4, Zsh и Ksh93.
- В Ksh93 массивы, типы которых не указаны явно, не обязательно индексируются. Массивы, определенные с использованием составных

присваиваний, которые задают индексы, по умолчанию являются ассоциативными. В Bash ассоциативные массивы могут быть созданы *только* путем явного объявления их как ассоциативных, в противном случае они всегда индексируются. Кроме того, ksh93 имеет несколько других составных структур, типы которых могут быть определены синтаксисом составного присваивания, используемым для их создания.

- В Ksh93 использование `=` составного оператора присваивания сбрасывает массив, включая любые атрибуты, которые были установлены в массиве до присвоения. Для сохранения атрибутов необходимо использовать `+=` оператор. Однако объявление ассоциативного массива, а затем попытка `a=(…)` сложного присваивания стиля без указания индексов является ошибкой. Я не могу объяснить это несоответствие.

```
 $ ksh -c'функция f { typeset -a a; a=([0]=foo [1]=bar); typeset
-p a; }; Атрибут f' # потерян, и поскольку заданы нижние индекс
ы, по умолчанию используется ассоциативный.
набор текста -A a=([0]=foo [1]=bar)
 $ ksh -c 'функция f { typeset -a a; a+=([0]=foo [1]=bar); types
et -p a; }; f' # Теперь использование += дает нам ожидаемые резу
льтаты.
набор текста -a a=(foo bar)
 Функция $ ksh -c ' function f { typeset -A a; a=(foo bar); type
set -p a; }; f' # Кроме того, обратная функция НЕ сбрасывает атр
ибут. Понятия не имею, почему.
 ksh: f: строка 1: невозможно добавить массив индексов к ассоциа
тивному массиву a
```

- Только Bash и mksh поддерживают составное присваивание со смешанными явными индексами и автоматически увеличивающимися индексами. В ksh93, чтобы указать отдельные индексы в составном присваивании, должны быть указаны все индексы (или ни одного). Zsh вообще не поддерживает указание отдельных индексов.
- Добавление к составному присваиванию - довольно удобный способ добавления элементов после последнего индекса массива. В Bash это также устанавливает режим добавления для всех отдельных назначений в составном назначении, так что, если указан нижний индекс, последующие элементы будут добавлены к предыдущим значениям. В ksh93 это приводит к игнорированию индексов, заставляя добавлять все после последнего элемента. (Добавление имеет другое значение из-за поддержки многомерных массивов и вложенных составных структур данных.)

```
 $ ksh -c 'функция f { typeset -a a; a+=(foo bar baz); a+=([3]=
blah [0]=bork [1]=blarg [2]=zooj); typeset -p a; }; f' # ksh93 з
аставляетдобавление к массиву без учета
набора индексов -a a=(foo bar baz '[3]=blah' '[0]=bork' '[1]= bl
arg' '[2]=zooj')
 $ bash -c 'функция f { typeset -a a; a+=(foo bar baz); a+=(blah
[0]=bork blarg zooj); typeset -p a; }; f' # Bash применяет += к
каждому отдельному индексу.
объявить -a a='([0]="foobork" [1]="barblarg" [2]= "bazzooj" [3]=
"бла")'
 $ mksh -c 'функция f { typeset -a a; a+=(foo bar baz); a+=(blah
[0]=bork blarg zooj); typeset -p a; }; f' # Mksh действительно п
охож на Bash, но удаляет предыдущие значения, а не добавляет.
набор -A
набор текста a [0] =
набор текста bork a [1] =
набор текста blarg a[2] =
набор текста zooj a [3] = бла
```

- В Bash и Zsh параметр назначения альтернативного значения expansion (`${arr[idx]:=foo}`) оценивает нижний индекс дважды, сначала, чтобы определить, следует ли расширять альтернативу, а затем, чтобы определить индекс, которому нужно назначить альтернативу. Смотрите порядок оценки.

```
 $ : ${_[$(echo $RANDOM >&2)1]:=$( echo привет >&2)}
13574
привет
14485
```

- В Zsh массивы индексируются начиная с 1 в режиме по умолчанию. Режимы эмуляции требуются для того, чтобы получить какую-либо переносимость.
- Zsh и mksh не поддерживают составные аргументы присваивания `typeset` .
- Ksh88 не поддерживает современный синтаксис назначения составных массивов. Оригинальный (и наиболее переносимый) способ назначения нескольких элементов - использовать `set -A name arg1 arg2 …` синтаксис. Это поддерживается почти всеми оболочками, которые поддерживают ksh-подобные массивы, за исключением Bash. Кроме того, эти оболочки обычно поддерживают необязательный `-s` аргумент `set` , который выполняет лексикографическую сортировку либо по элементам массива, либо по позиционным параметрам. Bash не имеет встроенной возможности сортировки, кроме обычных операторов сравнения.

```
 $ ksh -c 'set -A arr -- foo bar bork baz; typeset -p arr' # Кла
ссический синтаксис присваивания массива
typeset -a arr=(foo bar bork baz)
 $ ksh -c 'set -sA arr -- foo bar bork baz; typeset -p arr' # Co
бственная сортировка!
набор текста -a arr=(bar baz bork foo)
 $ mksh -c 'set -sA arr -- foo "[3]=bar" "[2]=baz" "[7]=bork"; t
ypeset -p arr' # Вероятно, ошибка. Я думаю, что сопровождающий з
нает об этом.
набор -
набор arr arr[2]=baz
набор текста arr[3]=
набор текста bar arr[7]=
набор текста bork arr[8]=foo
```

- Порядок оценки назначений, связанных с массивами, значительно варьируется в зависимости от контекста. Примечательно, что порядок вычисления нижнего индекса или значения в первую очередь может меняться практически в каждой оболочке как для расширений, так и для арифметических переменных. Подробности см. в разделе Порядок оценки.
- Bash 4.1. * и ниже не могут использовать отрицательные индексы для адресации индексов массива относительно индекса с наибольшим номером. Вы должны использовать расширение подстрочного индекса, то `"${arr[@]:(-n):1}"` есть для расширения n-го последнего элемента (или следующего по старшинству индексируемого после `n` того, как значение if `arr[n]` не задано). В Bash 4.2 вы можете расширять (но не присваивать) отрицательный индекс. В Bash 4.3, ksh93 и zsh вы можете как назначать, так и расширять отрицательные смещения.
- ksh93 также имеет дополнительную нотацию среза: `"${arr[n..m]}"` где `n` и `m` являются арифметическими выражениями. Они необходимы для использования с многомерными массивами.
- Присвоение или ссылка на отрицательные индексы в mksh приводит к обтеканию. Максимальный индекс, по-видимому `UINT_MAX` , равен , который будет адресован `arr[-1]` .
- Пока `-v var` что тест Bash не поддерживает отдельные индексы массива. Вы можете указать имя массива, чтобы проверить, определен ли массив, но не можете проверить элемент. ksh93 `-v` поддерживает оба. В других оболочках отсутствует `-v` тест.

Ошибки

- **Исправлено в 4.3** Bash 4.2. * и ранее учитывает каждый фрагмент составного присваивания, включая нижний индекс для глобулирования. Часть с нижним индексом считается заключенной в кавычки, но любые символы без кавычек в правой части будут `[…]=` объединены с нижним индексом и будут считаться глобусом. Поэтому вы должны заключать в кавычки все, что находится справа от `=` знака. Это исправлено в 4.3, так что каждый оператор присваивания нижнего индекса расширяется по тем же правилам, что и обычное присваивание. Это также корректно работает в ksh93.

```
$ touch '[1]=a'; bash -c 'a=([1]=*); echo "${a[@]}"'
[1]=a
```

mksh имеет аналогичную, но еще худшую проблему в том, что весь нижний индекс считается глобусом.

```
$ touch 1=a; mksh -c 'a=([123]=*); print -r -- "${a[@]}"'
1=a
```

- **Исправлено в 4.3** В дополнение к вышеупомянутой проблеме с глобализацией, назначения, предшествующие "объявлению", оказывают дополнительное влияние на расширение фигурных скобок и имени пути.

```
$ set -x; foo=объявление строки arr=( {1..10} )
+ foo=bar
+ объявить 'arr= (1)' 'arr= (2)' 'arr= (3)' 'arr= (4)' 'arr =
(5)' 'arr = (6)' 'arr= (7)' 'arr = (8)' 'arr=(9)' 'arr=(10)'

$ touch xy=foo
+ коснитесь xy=foo
$ объявить x[y]=*
+ объявить 'x[y]=*'
$ foo= объявление строки x[y]=*
+ foo=bar
+ объявить xy=foo
```

Каждое слово (все присвоение) подлежит глобализированию и расширению в фигурных скобках. Похоже, это запускает тот же странный режим расширения, `let` что и , `eval` , другие команды объявления, а может быть, и больше.

- **Исправлено в 4.3** Косвенность в сочетании с другим модификатором, расширяющим массивы до одного слова.

```
$ a=({a..c}) b=a[@]; printf '<%s> ' "${!b}"; echo; printf '<%s>
' "${!b/%/foo}"; echo
<a> <b> <c>
<a b cfoo>
```

- **Исправлено в 4.3**. Подстановки процессов оцениваются внутри индексов массива. Zsh и ksh не делают этого ни в каком арифметическом контексте.

```

# напечатать "moo"
dev=fd=1 _[1<(echo moo >&2)]=

# Форк-бомба
${dev[${dev='dev[1>(${dev[dev]})]'}]}
```

Порядок оценки

Вот некоторые неприятные детали порядка оценки назначения массива. Вы можете использовать этот тестовый код (https://gist.github.com/ormaaj/4942297) для генерации этих результатов.

```
Каждый тестовый набор выводит порядок оценки для контекстов назначени
я индексированного массива
. Каждый контекст проверяется на разложение (представленное цифрами)
и
арифметику (буквы), упорядоченные слева направо внутри выражения.
Результат соответствует способу переупорядочения оценки для каждой об
олочки:

a[ $1 a ]=${b[ $2 b]:=${c[ $3 c ]}} Нет атрибутов
a[ $ 1 a ] =${b[ $ 2 b]:=c[ $ 3 c ]} набор текста -ia a
a[ $ 1 a]=${b[ $ 2 b]:=c[ $ 3 c ]} набор текста -ia b
a[ $ 1a ]=${b[ $2 b ]:=c[ $3 c ]} набор текста -ia a b
(( a[ $1 a ] = b[ $2 b ] ${c [ $3 c ]} )) Нет атрибутов
(( a[ $ 1 a ] = ${b[ $ 2 b]:=c[ $3 c ]} )) набор текста -ia b
a+=( [ $1 a ] =${b[ $ 2 b]:=${c[ $ 3 c ]}} [ $ 4 d ] =$(( $ 5 e )) )
набор текста -a a
a+=( [ $ 1 a ] = ${b[ $ 2 b]:=c[ $ 3 c ]} [ $ 4 d ] = $ {5}e )набор т
екста -ia a

bash: 4.2.42(1) -релиз
2 b 3 c 2 b 1 a
2 b 3 2 b 1 a c
2 b 3 2 b c 1 a
2 b 3 2 b c 1 a c
1 2 3 c b a
1 2 b 3 2 b c c a
1 2 b 3 c 2 b 4 5 e a d
1 2 b 3 2 b 4 5 a c d e

ksh93: версия AJM 93v- 2013-02-22
1 2 b b a
1 2 b b a
1 2 b b a
1 2 b b a
1 2 3 c b a
1 2 b b a
1 2 b b a 4 5 e d
1 2 b b a 4 5 d e

mksh: @(#)MIRBSD KSH R44 2013/02/24
2 b 3 c 1 a
2 b 3 1 a c
2 b 3 c 1 a
2 b 3 c 1 a
1 2 3 c a b
1 2 b 3 c a
1 2 b 3 c 4 5 e a d
1 2 b 3 4 5 a c d e

zsh: 5.0.2
2 b 3 c 2 b 1 a
2 b 3 2 b 1 a c
2 b 1 a
2 b 1 a
1 2 3 c b a
```

```
1 2 b a
1 2 b 3 c 2 b 4 5 e
1 2 b 3 2 b 4 5
```

Смотрите также

- Расширение параметров (содержит разделы для массивов)
- Классический цикл for (содержит несколько примеров для перебора массивов)
- Команда declare builtin
- BashFAQ 005 - Как я могу использовать переменные массива? (http://mywiki.wooledge.org/BashFAQ/005)- Очень подробное обсуждение массивов со многими примерами.
- BashSheet - Массивы (http://mywiki.wooledge.org/BashSheet#Arrays) - краткий справочник по Bashsheet в вики Greycat.

Обсуждение

captainmish, 2011/05/26 10:16 ()

Удобно для "нарезки" - с

```
${МАССИВ[*]:N:M}
```

M подразумевается, если оно опущено, так что, например

```
${МАССИВ[*]:5}
```

будет расширяться до элементов от 6 до {конец массива}

zik, 2011/06/13 17:24 ()

Нет необходимости вычитать 1 из числа элементов. Просто измените условие завершения на '<'.

```
N_ELEMENTS=${#ПРЕДЛОЖЕНИЕ[@]}

для ((i = 0; i < N_ELEMENTS; i ++)); do # приращение равно i плюс
плюс, но предварительный просмотр не показывает этого.
 echo "Элемент $ i: '${ПРЕДЛОЖЕНИЕ [i]}'"
выполнено
```

Альтаир IV, 2012/01/12 13:25 ()

Вы можете безопасно перебирать разреженные и ассоциативные массивы, используя ${!шаблон расширения массива [@]}.

```
для i в "${!array[@]}"; выполнить
эхо "${array[$i]}"
Выполнено
```

Bash 4.2+ также имеет отрицательную индексацию массива. Например, вы можете расширить предпоследний элемент с помощью ${array[-2]} и даже одновременно применить к нему расширение параметра!

Клепт Немо, 2012/11/29 03:43 ()

Функция isSubset в разделе косвенности действительно могла бы использовать некоторое объяснение, в частности:

[1]

```
 локальный -a 'xkeys=("${!'"$1"'[@]}")' ' ключи=
("${!'"$2"'[@]}")'
```

Я не смог найти никаких примеров подобного использования local / declare где-либо еще. В BashFAQ [1] говорится: "правая часть присваивания не анализируется оболочкой". Тем не менее, очевидно - и также в следующем примере - выполняется некоторый уровень синтаксического анализа.

```
 локальный -a 'a=({0..5})'
```

Хотя я проверил себя и для правой части declare / eval не смог вызвать никаких нежелательных побочных эффектов или выполнения произвольного кода, я не могу гарантировать, что в этой форме любая функция безопасна. Кроме того, поведение, которое позволяет этим функциям разыменовывать косвенные ссылки на переменные, кажется недокументированным и работает только с параметром array . Следующее не приведет к расширению:

```
 локальные 'xkeys="${!'"$1"'[@]}"'
```

[2]

```
 установить -- "${@/%/[ключ]}"
```

Устанавливает позиционные параметры в значение расширения параметра.

[3]

```
 [[ ${!2+_} && ${!1} == ${!2} ]] || возврат 1
```

Ссылаясь на ${name+_}, цитируя руководство по bash: "Пропуск двоеточия приводит к тестированию только для параметра, который не задан. Другими словами, если включено двоеточие, оператор проверяет как существование

параметра, так и то, что его значение не равно null; если двоеточие опущено, оператор проверяет только на существование. "

[1] http://mywiki .wooledge.org/BashFAQ/006#Indirection (http://mywiki.wooledge.org/BashFAQ/006#Indirection)