Замена процесса

Подстановка процессов - это форма перенаправления, при которой входные или выходные данные процесса (некоторая последовательность команд) отображаются в виде временного файла.

```
<( <СПИСОК> )

>( <СПИСОК> )
```

Подстановка процессов выполняется **одновременно** с расширением параметров, заменой команд и арифметическим расширением.

Список команд `<LIST>` выполняется, и его

- стандартный выходной файловый дескриптор в `<( … )` виде или
- стандартный входной `>( … )` файловый дескриптор в виде

подключается к FIFO или файлу в `/dev/fd/` . Имя файла (к которому подключен filedescriptor) затем используется в качестве замены для `<(…)` -конструкции.

Это, например, позволяет передавать данные команде, которые не могут быть достигнуты с помощью конвейерной обработки (которая ожидает свои данные не из `stdin` файла, а из файла).

Область применения

Примечание: Согласно многочисленным комментариям и источникам, область файловых дескрипторов подстановки процессов **не** является стабильной, гарантированной или определенной bash. Более новые версии bash (5.0+), похоже, имеют более короткую область действия, а область подстановок, похоже, короче области действия функции. См. stackexchange (https://unix.stackexchange.com/questions/425456/conditional-process-substitution) и stackoverflow (https://stackoverflow.com/questions/46660020/bash-what-is-the-scope-of-the-process-substitution); последнее обсуждение содержит скрипт, который может проверять поведение области видимости в каждом конкретном случае
Если подстановка процесса расширяется в качестве аргумента функции, расширяется до переменной среды во время вызова функции или расширяется до любого назначения внутри функции, подстановка процесса будет "удерживаться открытой" для использования любой командой внутри функции или ее вызываемых, пока функция, в которой онабыл установлен возврат. Если в вызываемом объекте снова задана та же переменная, если только новая переменная не является локальной, предыдущая подстановка процесса закрывается и будет недоступна вызывающему объекту, когда вызываемый объект вернется.

По сути, подстановки процессов, расширенные до переменных внутри функций, остаются открытыми до тех пор, пока функция, в которой произошла подстановка процессов, не вернет - даже если ей присвоены локальные значения, заданные вызывающей функцией. Динамическая область видимости не защищает их от закрытия.

Примеры

Этот код бесполезен, но он демонстрирует, как он работает:

```
$ echo <(ls)
/dev/fd/63
```

Затем можно получить доступ к выходным `ls` данным -program, прочитав файл `/dev/fd/63` .

Рассмотрим следующее:

```
разница <(ls  "$first_directory") <(ls "$second_directory")
```

Это позволит сравнить содержимое каждого каталога. В этой команде каждый *процесс заменяется файлом*, и diff не видит <(bla) , он видит два файла, поэтому эффективная команда выглядит примерно так

```
/ diffdev/fd/63 /dev/fd/64
```

где эти файлы записываются и уничтожаются автоматически.

Избегание подоболочек

> Смотрите также: BashFAQ / 024 (http://mywiki.wooledge.org/BashFAQ/024) – *я устанавливаю переменные в цикле, который находится в конвейере. Почему они исчезают после завершения цикла? Или, почему я не могу передавать данные для чтения?*

Одно из наиболее распространенных применений подстановок процессов - избегать конечной подоболочки, возникающей в результате выполнения конвейера. Ниже приведен **неправильный** фрагмент кода для подсчета всех файлов в `/etc` :

```
счетчик =0

 найти /etc -print0 | в то время  как IFS= read -rd " _; do
    ((counter++))
done

 echo "$counter files" # выводит "0 файлов"
```

Из-за канала `while read; do … done` часть выполняется в подоболочке (в Bash, по умолчанию), что означает `counter` , что увеличивается только внутри подоболочки. Когда конвейер завершается, дочерняя оболочка завершается, а `counter` видимое значение по `echo` -прежнему равно "0"!

Замена процесса помогает нам избежать оператора канала (причина подоболочки):

```
счетчик =0

, в то время  как IFS= read -rN1 _; do
    ((counter++))
сделано < <(find /etc -printf ' ')

эхо  "$counter files"
```

Это обычное перенаправление входного файла `< FILE` , просто `FILE` в данном случае это результат подстановки процесса. Важно отметить, что пробел необходим для устранения неоднозначности синтаксиса из документов here .

```
: < <(КОМАНДА) # Хорошо.
: <<(...) # Неправильно. Будет проанализирован как heredoc. Bash заве
ршается с ошибкой, когда он сталкивается с метасимволом без кавычек "
("
: ><(...) # Технически корректный, но бессмысленный синтаксис. Bash о
ткрывает канал для записи, в то время как команды в процессе подстано
вки имеют свой стандартный вывод, подключенный к каналу.
```

Подстановка процесса, назначенная параметру

Этот пример демонстрирует, как можно сделать замены процессов похожими на "проходимые" объекты. Это приводит к преобразованию вывода `f` аргумента 's в верхний регистр.

```
f() {
    cat "$1" >"$x"
}

x=>(tr '[:нижний:]" [:верхний:]') f <(echo' hi там')
```

См. Раздел выше о области применения

Проблемы с ошибками и переносимостью

- Замена процесса не задается POSIX.
- Замена процесса полностью отключена в режиме Bash POSIX.
- Подстановка процессов осуществляется с помощью Bash, Zsh, Ksh{88,93}, но не (пока) производных pdksh (mksh). Вместо этого могут использоваться сопроцессы.
- Подстановка процессов поддерживается только в системах, которые поддерживают либо именованные каналы (FIFO - специальный файл), либо `/dev/fd/*` метод доступа к открытым файлам. Если система не поддерживает `/dev/fd/*` , Bash возвращается к созданию именованных каналов. Обратите внимание, что не все оболочки, поддерживающие замену процесса, имеют этот запасной вариант.
- Bash оценивает подстановки процессов в индексах массива, но не в других арифметических контекстах. Ksh и Zsh этого не делают. (Возможная ошибка)

```
dev
# print "moo"=fd=1 _[1<(echo moo >&2)] =
# форк-бомба
 ${dev[${dev='dev[1>(${dev[dev]})]'}]}
```

- Проблемы с ожиданием, условиями гонки и т. Д.: https://groups.google.com/forum/?fromgroups =#! тема/comp.unix.shell/GqLNzUA4ulA (https://groups.google.com/forum/?fromgroups=#!topic/comp.unix.shell/GqLNzUA4ulA)

Смотрите также

- Внутренний: введение в расширение и замену
- Внутренняя: Bash в дереве процессов (подоболочки)
- Внутренняя: перенаправление

Обсуждение

liungkejin, 2013/05/31 11:46 (), 2013/06/22 11:58 ()

Я нашел забавную вещь:

```
(эхо "ДА")> > (читать str; эхо "1: $ {str}:первый";)> > (читать s
str; эхо "2: $ sstr: два")> > (читать ssstr; эхо "3: $ ssstr: тр
и")
```

дает

```
1:2:3: ДА: три: два: первый
```