

## 10.3. Управление ходом выполнения цикла

### break, continue

Для управления ходом выполнения цикла служат команды **break** и **continue** [1] и точно соответствуют своим аналогам в других языках программирования. Команда **break** прерывает исполнение цикла, в то время как **continue** передает управление в начало цикла, минуя все последующие команды в теле цикла.

#### Пример 10-20. Команды break и continue в цикле

```
#!/bin/bash

LIMIT=19 # Верхний предел

echo
echo "Печать чисел от 1 до 20 (исключая 3 и 11)."

a=0

while [ $a -le "$LIMIT" ]
do
    a=$((a+1))

    if [ "$a" -eq 3 ] || [ "$a" -eq 11 ] # Исключить 3 и 11
    then
        continue # Переход в начало цикла.
    fi

    echo -n "$a "
done

# Упражнение:
# Почему число 20 тоже выводится?

echo; echo

echo Печать чисел от 1 до 20, но взгляните, что происходит после
вывода числа 2

#####
#

# Тот же цикл, только 'continue' заменено на 'break'.

a=0

while [ "$a" -le "$LIMIT" ]
do
    a=$((a+1))

    if [ "$a" -gt 2 ]
```

```

then
    break # Завершение работы цикла.
fi

echo -n "$a "
done

echo; echo; echo

exit 0

```

Команде **break** может быть передан необязательный параметр. Команда **break** без параметра прерывает тот цикл, в который она вставлена, а **break N** прерывает цикл, стоящий на N уровней выше (причем 1-й уровень -- это уровень текущего цикла, прим. перев.).

### Пример 10-21. Прерывание многоуровневых циклов

```

#!/bin/bash
# break-levels.sh: Прерывание циклов.

# "break N" прерывает исполнение цикла, стоящего на N уровней
# выше текущего.

for outerloop in 1 2 3 4 5
do
    echo -n "Группа $outerloop:  "

    for innerloop in 1 2 3 4 5
    do
        echo -n "$innerloop "

        if [ "$innerloop" -eq 3 ]
        then
            break # Попробуйте "break 2",
                  # тогда будут прерываться как вложенный, так и
внешний циклы
        fi
    done

    echo
done

echo

exit 0

```

Команда **continue**, как и команда **break**, может иметь необязательный параметр. В простейшем случае, команда **continue** передает управление в начало текущего цикла, а команда **continue N** прерывает исполнение текущего цикла и передает управление в начало внешнего цикла, отстоящего от текущего на N уровней (причем 1-й уровень -- это уровень текущего цикла, прим. перев.).

### Пример 10-22. Передача управление в начало внешнего цикла

```
#!/bin/bash
# Команда "continue N" передает управление в начало внешнего
цикла, отстоящего от текущего на N уровней.

for outer in I II III IV V          # внешний цикл
do
    echo; echo -n "Группа $outer: "

    for inner in 1 2 3 4 5 6 7 8 9 10 # вложенный цикл
    do

        if [ "$inner" -eq 7 ]
        then
            continue 2 # Передача управления в начало цикла 2-го
уровня.                # попробуйте убрать параметр 2 команды
"continue"
        fi

        echo -n "$inner " # 8 9 10 никогда не будут напечатаны.
    done

done

echo; echo

# Упражнение:
# Подумайте, где реально можно использовать "continue N" в
сценариях.

exit 0
```

### Пример 10-23. Живой пример использования "continue N"

```
# Albert Reiner привел пример использования "continue N":
# -----

# Допустим, у меня есть большое количество задач, обрабатывающие
некоторые данные,
#+ которые хранятся в некоторых файлах, с именами, задаваемыми по
шаблону,
#+ в заданном каталоге.
#+ Есть несколько машин, которым открыт доступ к этому каталогу
#+ и я хочу распределить обработку информации между машинами.
#+ тогда я обычно для каждой машины пишу нечто подобное:

while true
do
    for n in .iso.*
    do
        [ "$n" = ".iso.opts" ] && continue
        beta=${n#.iso.}
        [ -r .Iso.$beta ] && continue
        [ -r .lock.$beta ] && sleep 10 && continue
        lockfile -r0 .lock.$beta || continue
        echo -n "$beta: " `date`
        run-isotherm $beta
        date
```

```


ls -alF .Iso.$beta
[ -r .Iso.$beta ] && rm -f .lock.$beta
continue 2
done
break
done

# Конкретная реализация цикла, особенно sleep N, зависит от
конкретных применений,
#+ но в общем случае он строится по такой схеме:

while true
do
  for job in {шаблон}
  do
    {файл уже обработан или обрабатывается} && continue
    {пометить файл как обрабатываемый, обработать, пометить как
    обработанный}
    continue 2
  done
  break          # Или что нибудь подобное `sleep 600', чтобы
избежать завершения.
done

# Этот сценарий завершит работу после того как все данные будут
обработаны
#+ (включая данные, которые поступили во время обработки).
Использование
#+ соответствующих lock-файлов позволяет вести обработку на
нескольких машинах
#+ одновременно, не производя дублирующих вычислений [которые, в
моем случае,
#+ выполняются в течении нескольких часов, так что для меня это
очень важно].
#+ Кроме того, поскольку поиск необработанных файлов всегда
начинается с
#+ самого начала, можно задавать приоритеты в именах файлов.
Конечно, можно
#+ обойтись и без `continue 2', но тогда придется ввести
дополнительную
#+ проверку -- действительно ли был обработан тот или иной файл
#+ (чтобы перейти к поиску следующего необработанного файла).

```

 Конструкция **continue N** довольно сложна в понимании и применении, поэтому, вероятно лучше будет постараться избегать ее использования.

## Примечания

- [1] Эти команды являются [встроенными командами](#) языка сценариев командной оболочки (shell), в то время как [while](#), [case](#) и т.п. -- являются [зарезервированными словами](#).

