



Факультет
биоинженерии и биоинформатики
Московский государственный университет имени М.В.Ломоносова



BASH



- ▶ Перенаправления
- ▶ Подстановки
- ▶ Функции
- ▶ Логические операции
- ▶ Управление задачами

Взаимодействие с программой



Как передать информацию:

- ▶ аргументы командной строки
- ▶ стандартный поток ввода (`STDIN`)
- ▶ переменные окружения
- ▶ сигналы

Как получить информацию:

- ▶ стандартные потоки вывода (`STDOUT` , `STDERR`)
- ▶ код возврата



Перенаправления

Файловые дескрипторы



Файловый дескриптор – это идентификатор (целое число) потока ввода/вывода

Удобно считать его "ссылкой" на объект, который будет использоваться для операций ввода/вывода

Всегда существуют ссылки 0, 1 и 2 – стандартные потоки ввода/вывода; на что они указывают определяет родительский процесс (например, bash)

Файловый ввод/вывод



Открыть file на чтение, прикрепить к нему ссылку с номером n

```
$ command [n]<file
```

Открыть file на запись, прикрепить к нему ссылку с номером n

```
$ command [n]>file
```

Открыть file на дозапись, прикрепить к нему ссылку с номером n

```
$ command [n]>>file
```

Открыть file на чтение и запись, прикрепить ссылку с номером n

```
$ command [n]<>file
```

Конвейеры



Для работы конвейера требуется канал (pipe) – область оперативной памяти, к которой можно прикрепить два файловых дескриптора, по одному для записи и чтения

При интерпретации `cmd1 | cmd2`

- ▶ `bash` создает "безымянный" канал (не путать с именованными каналами)
- ▶ запускает два дочерних процесса
 - ▶ в одном подключает `fd1` к каналу на запись и запускает `cmd1`
 - ▶ в другом подключает `fd0` к каналу на чтение и запускает `cmd2`
- ▶ ждет завершения обоих процессов

Дубликация файловых дескрипторов



Прикрепить ссылку `n` к объекту, на который указывает `m`, объект должен быть доступен для чтения

```
$ command [n]<&m
```

Аналогично для записи

```
$ command [n]>&m
```

Закреть файловый дескриптор с номером `n`, открытый на чтение, т.е. "удалить ссылку `n`"

```
$ command [n]<&-
```

Аналогично для записи

```
$ command [n]>&-
```

Обычно закрывать дескрипторы вручную не требуется

Перенаправление с объединением потоков



```
$ command &>file
```

равнозначно

```
$ command >file 2>&1 # порядок важен!
```

```
$ command &>>file
```

равнозначно

```
$ command >>file 2>&1 # порядок важен!
```

```
$ cmd1 |& cmd2
```

равнозначно

```
$ cmd1 2>&1 | cmd2
```

Here Documents и Here Strings



подать строки в качестве потока ввода с номером n

```
$ command [n]<<METKA
```

```
line1
```

```
line2
```

```
line3
```

```
...
```

```
METKA
```

подать слово в качестве потока ввода

```
$ command [n]<<<word
```

Строки и слово могут претерпевать некоторые подстановки!



Подстановки

Виды подстановок



- ▶ раскрытие фигурных скобок (brace expansion)
- ▶ подстановка тильды (tilde expansion)
- ▶ подстановка переменных (parameter expansion)
- ▶ подстановка команд (command substitution)
- ▶ арифметические подстановки (arithmetic expansion)
- ▶ подстановка процессов (process substitution)
- ▶ разбиение на аргументы (word splitting)
- ▶ интерпретация файловых масок (pathname expansion)
- ▶ удаление экранирующих символов (quote removal)

Раскрытие фигурных скобок



```
$ echo A{B,C}
```

```
AB AC
```

```
$ echo {tmp{1,2},file}.txt
```

```
tmp1.txt tmp2.txt file.txt
```

```
$ echo {k..n}
```

```
k l m n
```

```
$ echo {7..13} {Γ..Ж}
```

```
7 8 9 10 11 12 13 {Γ..Ж}
```

```
$ echo a{7..12..4}.tmp b{B..Z..12}.tmp
```

```
a7.tmp a11.tmp bB.tmp bN.tmp bZ.tmp
```

```
$ echo X{009..11}.tmp
```

```
X009.tmp X010.tmp X011.tmp
```

Подстановка переменных



```
$ value='*'
$ echo $value "$value"
a b c *
$ echo "$value1" "${value}1"
*1
$ echo "${vaie}1" "${vaie:-Q}1"
1 Q1
$ echo "${value:? 'ошибка'}"
*
$ echo "${value:? 'ошибка'}" "${vaie:? 'ошибка'}"
bash: vaie: ошибка

$ file='file-name.tar.gz'
$ echo "${file}/-name/0}" "${file}/-name}" "${file//[ae]/X}"
file0.tar.gz file.tar.gz filX-nXmX.tXr.gz
$ echo "${file%.*gz}" "${file%%.*gz}" "${file##*a}" "${file###*a}"
file-name.tar file-name me.tar.gz r.gz
```

... И ЕЩЕ МНОГО ВСЕГО: `man bash -> EXPANSION -> Parameter Expansion`

Подстановка команд



устаревший синтаксис

```
$ echo `ls -l`  
a b c
```

современный синтаксис

```
$ echo $(ls -l)  
a b c
```

экранирование подставляемого значения

```
$ echo "$(ls -l)"  
a  
b  
c
```

допустимы вложенные подстановки

```
$ getent group "$(id -g "$(whoami)")"  
user:x:1000:
```

Подстановка процессов

Позволяет подставить STDIN или STDOUT произвольной команды в виде файлоподобного объекта

```
$ cat <(id -un) <(pwd)
user
/home/user
$ echo {a..z} | tr ' ' '\n' | paste - <(seq 26) <(seq 26 -1 1) | grep '[num]'
m      13      14
n      14      13
u      21       6
$ getent group | cut -f4 -d: | tee \
>(grep -c '^$' | sed 's/^/empty:/') \
>(grep -Fc ',' | sed 's/^/multi:/') \
>(wc -l | sed 's/^/total:/') \
>'/dev/null'; sleep '0.1'
total:67
multi:2
empty:56
```

Как это работает?

```
$ echo <(ls -l)
/dev/fd/63
$ file <(ls -l)
/dev/fd/63: symbolic link to pipe:[279540]
$ ls -l /dev/fd
lrwxrwxrwx 1 root root 13 Nov 20 10:20 /dev/fd -> /proc/self/fd
```



Команды

Выполнение простых команд



- ▶ разбиение на аргументы
- ▶ отделение присвоений переменных и перенаправлений
- ▶ подстановки в оставшихся аргументах
- ▶ первый аргумент (при наличии) считается именем команды
- ▶ если имя не содержит слешей, то (до первого совпадения):
 - ▶ [при выполнении ряда условий] происходит подстановка алиасов
 - ▶ проверяется наличие одноименной функции
 - ▶ проверяется наличие одноименной встроенной команды (builtin)
 - ▶ проверяется наличие записи в хэш-таблице путей
 - ▶ производится поиск исполнимого файла в папках из переменной PATH
- ▶ выполняется функция, встроенная команда или файл

Алиасы

- ▶ проверяется только имя команды (есть исключение)
- ▶ не обрабатываются в неинтерактивном режиме (можно активировать с помощью `shopt`)
- ▶ имя не должно содержать ряда спец. символов, включая экранирующие
- ▶ обрабатываются рекурсивно, но каждая подстановка срабатывает один раз
- ▶ есть ряд других не самых очевидных ограничений

```
$ alias e='echo Numbers:' # shell builtin
```

```
$ e 1 2 3
```

```
Numbers: 1 2 3
```

```
$ 'e' 1 2 3
```

```
bash: e: command not found
```

```
$ unalias e # shell builtin
```

```
$ alias
```

```
alias grep='grep --color=auto'
```

```
alias ll='ls -l'
```

```
alias ls='ls --color=auto'
```

Функции



упрощенная запись

```
$ fname() { command1; command2; ...; }
```

внутри тела функции доступны позиционные аргументы

```
$ myfun() { echo "$1" | tr '[:lower:]' '[:upper:]'; }
```

```
$ myfun first second
```

```
FIRST
```

для манипуляций с определениями функций есть специальные опции

у встроенных команд `declare` и `unset`

```
$ declare -pf myfun
```

```
myfun ()
```

```
{
```

```
    echo "$1" | tr '[:lower:]' '[:upper:]'
```

```
}
```

```
$ unset -f myfun
```

```
$ declare -pf myfun
```

```
bash: declare: myfun: not found
```



Логические операции

Код возврата

- ▶ каждая команда возвращает статус завершения – целое число [0, 255]
- ▶ 0 означает успешное завершение, соответствует ИСТИНЕ в логических операциях
- ▶ остальные коды соответствуют различным ошибкам, воспринимаются как ЛОЖЬ
- ▶ некоторые коды больше 125 использует bash:
 - 126 команде соответствует файл, но нет права на его исполнение
 - 127 команда не найдена
 - 128+N выполнение команды было прервано сигналом с номером N

Код возврата каждой команды сохраняется в специальную переменную ?

```
$ sleep 10; echo $?
```

```
0
```

```
$ sleep 10
```

```
^C
```

```
$ echo $?
```

```
130
```

AND-списки и OR-списки команд



AND-список

```
$ cmd1 && cmd2
```

cmd2 выполняется только в случае успешного выполнения cmd1

```
$ true && echo OK; echo next
```

OK

next

```
$ false && echo OK; echo next
```

next

OR-список

```
$ cmd1 || cmd2
```

cmd2 выполняется только в случае ошибки выполнения cmd1

```
$ true || echo BAD; echo next
```

next

```
$ false || echo BAD; echo next
```

BAD

next

Циклы while и until



итерации продолжаются, пока cmd0 возвращает 0

```
$ while cmd0; do cmd1; cmd2; ...; done
```

наоборот, выполняется, пока cmd0 не вернет 0

```
$ until cmd0; do cmd1; cmd2; ...; done
```

бесконечный цикл, если в теле цикла
нет вызова встроенной команды break

```
$ while true; do cmd1; cmd2; ...; done
```

while часто используют со встроенной командой read, которая
читает строки из STDIN и записывает аргументы в переменные

```
$ echo -e '1 2 3\n4 5' | while read a b; do echo "a='${a}' b='${b}'; done
```

```
a='1' b='2 3'
```

```
a='4' b='5'
```

Конструкция if

упрощенная запись

```
$ if cmdA; then cmd0; [ elif cmdB; then cmd1; ] ... [ else cmd2; ] fi
```

В качестве cmdAB.. часто используют:

- ▶ конструкцию [[логическое выражение]]
- ▶ конструкцию ((арифметическое выражение))
- ▶ встроенную команду [
- ▶ программу [

НО МОЖНО И ТАК

```
$ echo word | if grep -q 'word'; then echo OK; fi
```

OK

```
$ echo WORD | if grep -q 'word'; then echo OK; fi
```

```
$ echo WORD | if grep -iq 'word'; then echo OK; fi
```

OK



Управление задачами

Активный и фоновый режимы

Bash содержит средства для управления выполнением задач

- ▶ Задача bash – это группа процессов, составляющих один конвейер
- ▶ Каждой задаче присваивается номер, начиная с 1
- ▶ Кроме активной задачи может быть запущено любое количество задач в фоновом режиме
- ▶ Процессам в фоновом режиме запрещено читать данные из терминала
- ▶ Конвейер будет запущен в фоновом режиме, если завершается символом &

выполняется в активном режиме (foreground)

```
$ sleep 3 # придется ждать 3 секунды
```

запустится в фоновом режиме (background)

```
$ sleep 3 & # можно сразу выполнять другие команды
```

```
[1] 11994
```

```
$ echo word
```

```
word
```

```
[1]+ Done
```

```
sleep 3
```

Смена режима выполнения задачи



посмотреть список задач

```
$ jobs
```

```
[1] Done           sleep 5 &  
[2]- Running      sleep 15 &  
[3]+ Stopped      nano 1.txt
```

перевести задачу в активный режим

```
$ fg %2
```

```
sleep 15
```

```
^Z
```

```
$ jobs
```

```
[2]- Stopped      sleep 15  
[3]+ Stopped      nano 1.txt
```

продолжить приостановленную (Ctrl+Z) задачу в фоне

```
$ bg %3
```

```
-- отображение "окна" nano
```

Принудительное завершение задачи



прервать активную задачу (послать SIGINT)

```
$ sleep 10
```

```
^C
```

остановить фоновую задачу (послать SIGTERM)

```
$ kill %2
```

```
[2]+  Terminated                  sleep 10
```

можно посылать другие сигналы

```
$ kill -9 %1
```

```
[2]+  Killed                        sleep 10
```

disown и nohup



Если команды выполняются удаленно, и происходит разрыв соединения, то `bash` получает от псевдотерминала сигнал `SIGHUP` и пересылает его всем процессам из таблицы задач

Чтобы удалить задачу из таблицы (без прерывания выполнения) можно использовать встроенную команду `disown %jobspec`

Можно запустить процесс, игнорирующий `SIGHUP`, с помощью программы `nohup`

Все приостановленные задачи будут принудительно завершены при завершении процесса `bash`