Bash-скрипты, часть 10: практические примеры

<https://likegeeks.com/write-shell-script/>

* [Блог компании RUVDS.com](https://habr.com/company/ruvds/),
* [Настройка Linux](https://habr.com/hub/linux/),
* [Системное администрирование](https://habr.com/hub/sys_admin/)
* Перевод

[Bash-скрипты: начало](https://habrahabr.ru/company/ruvds/blog/325522/)  
[Bash-скрипты, часть 2: циклы](https://habrahabr.ru/company/ruvds/blog/325928/)  
[Bash-скрипты, часть 3: параметры и ключи командной строки](https://habrahabr.ru/company/ruvds/blog/326328/)  
[Bash-скрипты, часть 4: ввод и вывод](https://habrahabr.ru/company/ruvds/blog/326594/)  
[Bash-скрипты, часть 5: сигналы, фоновые задачи, управление сценариями](https://habrahabr.ru/company/ruvds/blog/326826/)  
[Bash-скрипты, часть 6: функции и разработка библиотек](https://habrahabr.ru/company/ruvds/blog/327248/)  
[Bash-скрипты, часть 7: sed и обработка текстов](https://habrahabr.ru/company/ruvds/blog/327530/)  
[Bash-скрипты, часть 8: язык обработки данных awk](https://habrahabr.ru/company/ruvds/blog/327754/)  
[Bash-скрипты, часть 9: регулярные выражения](https://habrahabr.ru/company/ruvds/blog/327896/)  
[Bash-скрипты, часть 10: практические примеры](https://habrahabr.ru/company/ruvds/blog/328346/)  
[Bash-скрипты, часть 11: expect и автоматизация интерактивных утилит](https://habrahabr.ru/company/ruvds/blog/328436/)

В предыдущих материалах мы обсуждали различные аспекты разработки bash-скриптов, говорили о полезных инструментах, но до сих пор рассматривали лишь небольшие фрагменты кода. Пришло время более масштабных проектов. А именно, здесь вы найдёте два примера. Первый — скрипт для отправки сообщений, второй пример — скрипт, выводящий сведения об использовании дискового пространства.  
  
[](https://habrahabr.ru/company/ruvds/blog/328346/)  
  
Главная ценность этих примеров для тех, кто изучает bash, заключается в методике разработки. Когда перед программистом встаёт задача по автоматизации чего бы то ни было, его путь редко бывает прямым и быстрым. Задачу надо разбить на части, найти средства решения каждой из подзадач, а потом собрать из частей готовое решение.  
  
[https://habrastorage.org/files/1ba/550/d25/1ba550d25e8846ce8805de564da6aa63.png](https://ruvds.com/ru-rub/#order)

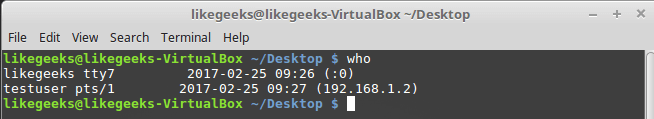
Отправка сообщений в терминал пользователя

В наши дни редко кто прибегает к одной из возможностей Linux, которая позволяет общаться, отправляя сообщения в терминалы пользователей, вошедших в систему. Сама по себе команда отправки сообщений, write, довольно проста. Для того, чтобы ей воспользоваться, достаточно знать имя пользователя и имя его терминала. Однако, для успешной отправки сообщения, помимо актуальных данных о пользователе и терминале, надо знать, вошёл ли пользователь в систему, не запретил ли он запись в свой терминал. В результате, перед отправкой сообщения нужно выполнить несколько проверок.  
  
Как видите, задача: «отправить сообщение», при ближайшем рассмотрении, оказалась задачей: «проверить возможность отправки сообщения, и, если нет препятствий, отправить его». Займёмся решением задачи, то есть — разработкой bash-скрипта.

▍Команды who и mesg

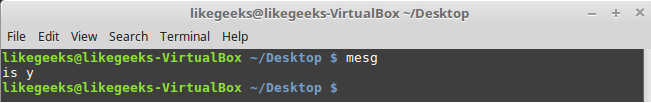
Ядром скрипта являются несколько команд, которые мы ещё не обсуждали. Всё остальное должно быть вам знакомо по предыдущим материалам.  
  
Первое, что нам тут понадобится — команда who. Она позволяет узнать сведения о пользователях, работающих в системе. В простейшем виде её вызов выглядит так:

$ who



*Результаты вызова команды who*  
  
В каждой строчке, которую выводит команда who, нас интересуют первых два показателя — имя пользователя и сведения о его терминале.  
  
По умолчанию запись в терминал разрешена, но пользователь может, с помощью команды mesg, запретить отправку ему сообщений. Таким образом, прежде чем пытаться что-то кому-то отправить, неплохо будет проверить, разрешена ли отправка сообщений. Если нужно узнать собственный статус, достаточно ввести команду mesg без параметров:

$ mesg

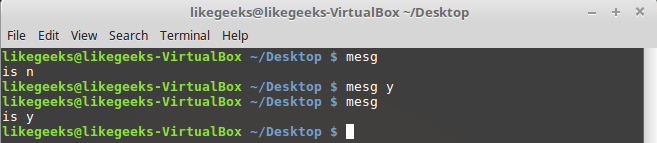


*Команда mesg*  
  
В данном случае команда вывела «is y», это значит, что пользователь, под которым мы работаем в системе, может принимать сообщения, отправленные в его терминал. В противном случае mesgвыведет «is n».  
  
Для проверки того, разрешена ли отправка сообщений какому-то другому пользователю, можно использовать уже знакомую вам команду who с ключом -T:

$ who -T

При этом проверка возможна только для пользователей, которые вошли в систему. Если такая команда, после имени пользователя, выведет чёрточку (-), это означает, что пользователь запретил запись в свой терминал, то есть, сообщения ему отправлять нельзя. О том, что пользователю можно отправлять сообщения, говорит знак «плюс» (+).  
  
Если у вас приём сообщений отключён, а вы хотите позволить другим пользователям отправлять вам сообщения, можно воспользоваться такой командой:

$ mesg y

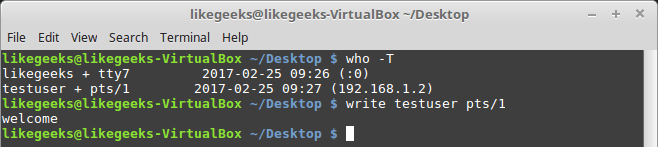


*Включение приёма сообщений от других пользователей*  
  
После включения приёма сообщений mesg возвращает «is y».  
Конечно, для обмена сообщениями нужны два пользователя, поэтому мы, после обычного входа в систему, подключились к компьютеру по ssh. Теперь можно поэкспериментировать.

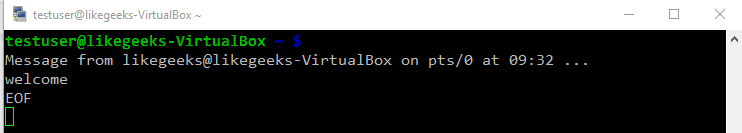
▍Команда write

Основной инструмент для обмена сообщениями между пользователями, вошедшими в систему — команда write. Если приём сообщений у пользователя разрешён, с помощью этой команды ему можно отправлять сообщения, используя его имя и сведения о терминале.  
  
Обратите внимание на то, что с помощью write можно отправлять сообщения пользователям, вошедшим в виртуальную консоль. Пользователи, которые работают в графическом окружении (KDE, Gnome, Cinnamon, и так далее), не могут получать подобные сообщения.  
  
Итак, мы, работая под пользователем likegeeks, инициируем сеанс связи с пользователем testuser, который работает в терминале pts/1, следующим образом:

$ write testuser pts/1



*Проверка возможности отправки сообщений и отправка сообщения*  
  
После выполнения вышеуказанной команды перед нами окажется пустая строка, в которую нужно ввести первую строку сообщения. Нажав клавишу ENTER, мы можем ввести следующую строку сообщения. После того, как ввод текста завершён, окончить сеанс связи можно, воспользовавшись комбинацией клавиш CTRL + D, которая позволяет ввести [символ конца файла](https://habrahabr.ru/company/ruvds/blog/326826/).  
  
Вот что увидит в своём терминале пользователь, которому мы отправили сообщение.



*Новое сообщение, пришедшее в терминал*  
  
Получатель может понять от кого пришло сообщение, увидеть время, когда оно было отправлено. Обратите внимание на признак конца файла, EOF, расположенный в нижней части окна терминала. Он указывает на окончание текста сообщения.  
  
Полагаем, теперь у нас есть всё необходимое для того, чтобы автоматизировать отправку сообщений с помощью сценария командной строки.

▍Создание скрипта для отправки сообщений

Прежде чем заниматься отправкой сообщений, нужно определить, вошёл ли интересующий нас пользователь в систему. Сделать это можно с помощью такой команды:

logged\_on=$(who | grep -i -m 1 $1 | awk '{print $1}')

Здесь результаты работы команды who передаются команде grep. Ключ -i этой команды позволяет игнорировать регистр символов. Ключ -m 1 включён в вызов команды на тот случай, если пользователь вошёл в систему несколько раз. Эта команда либо не выведет ничего, либо выведет имя пользователя (его мы укажем при вызове скрипта, оно попадёт в позиционную переменную $1), соответствующее первому найденному сеансу. Вывод grep мы передаём awk. Эта команда, опять же, либо не выведет ничего, либо выведет элемент, записанный в собственную переменную $1, то есть — имя пользователя. В итоге то, что получилось, попадает в переменную logged\_on.  
  
Теперь надо проверить переменную logged\_on, посмотреть, есть ли в ней что-нибудь:

if [ -z $logged\_on ]

then

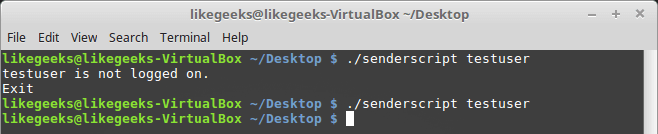
echo "$1 is not logged on."

echo "Exit"

exit

fi

Если вы не вполне уверенно чувствуете себя, работая с конструкцией if, взгляните на [этот](https://habrahabr.ru/company/ruvds/blog/325522/) материал.  
Скрипт, содержащий вышеописанный код, сохраним в файле senderscript и вызовем, передав ему, в качестве параметра командной строки, имя пользователя testuser.



*Проверка статуса пользователя*  
  
Тут мы проверяем, является ли logged\_on переменной с нулевой длиной. Если это так, нам сообщат о том, что в данный момент пользователь в систему не вошёл и скрипт завершит работу с помощью команды exit. В противном случае выполнение скрипта продолжится.

▍Проверка возможности записи в терминал пользователя

Теперь надо проверить, принимает ли пользователь сообщения. Для этого понадобится такая конструкция, похожая на ту, которую мы использовали выше:

allowed=$(who -T | grep -i -m 1 $1 | awk '{print $2}')

if [ $allowed != "+" ]

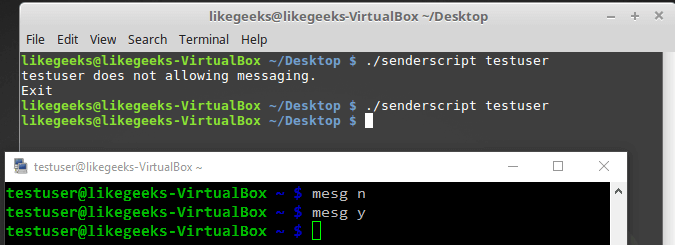
then

echo "$1 does not allowing messaging."

echo "Exit"

exit

fi



*Проверка возможности отправки сообщений пользователю*  
  
Сначала мы вызываем команду who с ключом -T. В строке сведений о пользователе, который может принимать сообщения, окажется знак «плюс» (+), если же пользователь принимать сообщения не может — там будет чёрточка (-). То, что получилось после вызова who, передаётся grep, а потом — awk, формируя переменную allowed.  
  
Далее, используя условный оператор, мы проверяем то, что оказалось в переменной allowed. Если знака «плюс» в ней нет, сообщим о том, что отправка сообщений пользователю запрещена и завершим работу. В противном случае выполнение сценария продолжится.

▍Проверка правильности вызова скрипта

Первым параметром скрипта является имя пользователя, которому мы хотим отправить сообщение. Вторым — текст сообщения, в данном случае — состоящий из одного слова. Для того, чтобы проверить, передано ли скрипту сообщение для отправки, воспользуемся таким кодом:

if [ -z $2 ]

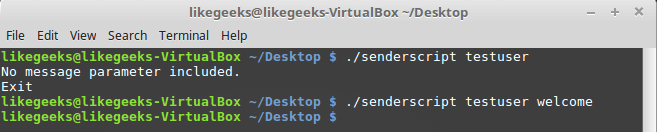
then

echo "No message parameter included."

echo "Exit"

exit

fi



*Проверка параметров командной строки, указанных при вызове скрипта*  
  
Тут, если при вызове скрипта ему не было передано сообщение для отправки, мы сообщаем об этом и завершаем работу. В противном случае — идём дальше.

▍Получение сведений о терминале пользователя

Прежде чем отправить пользователю сообщение, нужно получить сведения о терминале, в котором он работает и сохранить имя терминала в переменной. Делается это так:

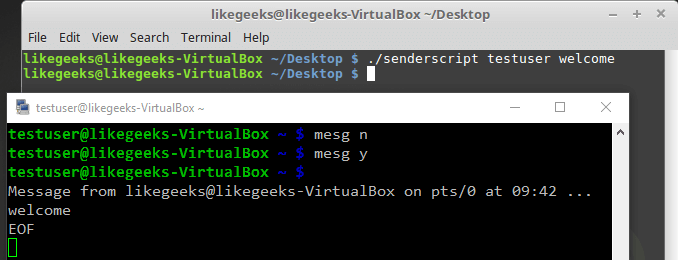
terminal=$(who | grep -i -m 1 $1 | awk '{print $2}')

Теперь, после того, как все необходимые данные собраны, осталось лишь отправить сообщение:

echo $2 | write $logged\_on $terminal

Вызов готового скрипта выглядит так:

$ ./senderscript testuser welcome

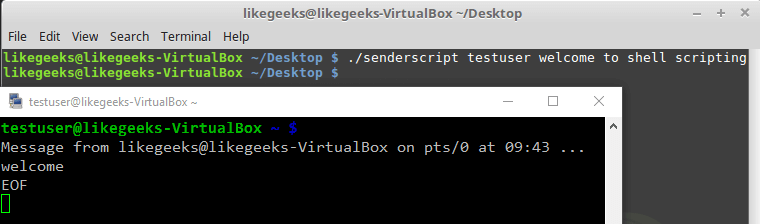


*Успешная отправка сообщения с помощью bash-скрипта*  
  
Как видно, всё работает как надо. Однако, с помощью такого сценария можно отправлять лишь сообщения, состоящие из одного слова. Хорошо бы получить возможность отправлять более длинные сообщения.

▍Отправка длинных сообщений

Попробуем вызвать сценарий senderscript, передав ему сообщение, состоящее из нескольких слов:

$ ./senderscript likegeeks welcome to shell scripting



*Попытка отправки длинного сообщения*  
  
Как видно, отправлено было лишь первое слово. Всё дело в том, что каждое слово сообщения воспринимается внутри скрипта как отдельная позиционная переменная. Для того, чтобы получить возможность отправки длинных сообщений, обработаем параметры командной строки, переданные сценарию, воспользовавшись [командой](https://habrahabr.ru/company/ruvds/blog/326328/) shift и циклом while.

shift

while [ -n "$1" ]

do

whole\_message=$whole\_message' '$1

shift

done

После этого, в команде отправки сообщения, воспользуемся, вместо применяемой ранее позиционной переменной $2, переменной whole\_message:

echo $whole\_message | write $logged\_on $terminal

Вот полный текст сценария:

#!/bin/bash

logged\_on=$(who | grep -i -m 1 $1 | awk '{print $1}')

if [ -z $logged\_on ]

then

echo "$1 is not logged on."

echo "Exit"

exit

fi

allowed=$(who -T | grep -i -m 1 $1 | awk '{print $2}')

if [ $allowed != "+" ]

then

echo "$1 does not allowing messaging."

echo "Exit"

exit

fi

if [ -z $2 ]

then

echo "No message parameter included."

echo "Exit"

exit

fi

terminal=$(who | grep -i -m 1 $1 | awk '{print $2}')

shift

while [ -n "$1" ]

do

whole\_message=$whole\_message' '$1

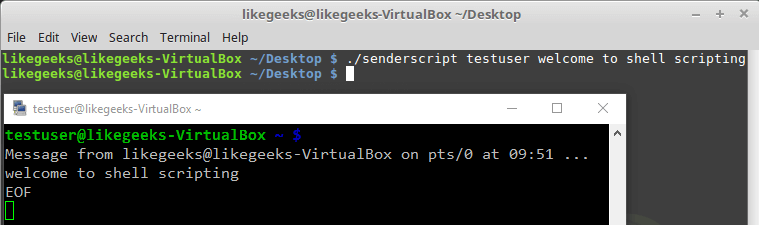
shift

done

echo $whole\_message | write $logged\_on $terminal

Испытаем его:

$ ./senderscript likegeeks welcome to shell scripting



*Успешная отправка длинного сообщения:*  
  
Длинное сообщение успешно дошло до адресата. Теперь рассмотрим следующий пример.

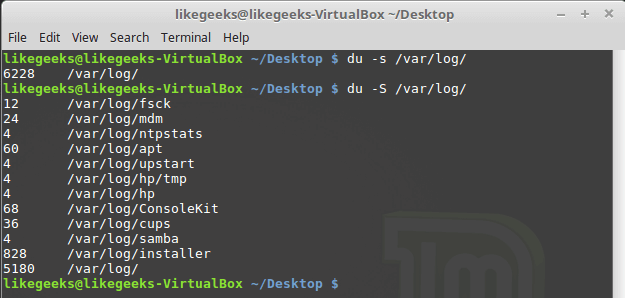
Скрипт для мониторинга дискового пространства

Сейчас мы собираемся создать сценарий командной строки, который предназначен для поиска в заданных директориях первой десятки папок, на которые приходится больше всего дискового пространства. В этом нам поможет [команда](https://likegeeks.com/main-linux-commands-easy-guide/#du-Command) du, которая выводит сведения о том, сколько места на диске занимают файлы и папки. По умолчанию она выводит сведения лишь о директориях, с ключом -aв отчёт попадают и отдельные файлы. Её ключ -s позволяет вывести сведения о размерах директорий. Эта команда позволяет, например, узнать объём дискового пространства, который занимают данные некоего пользователя. Вот как выглядит вызов этой команды:

$ du -s /var/log/

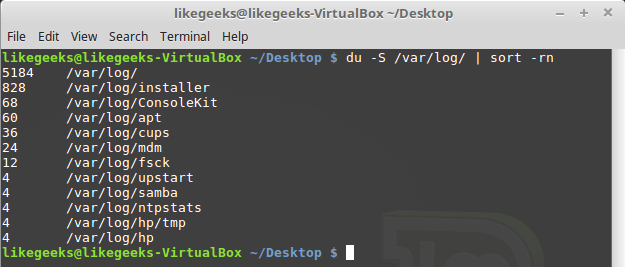
Для наших целей лучше подойдёт ключ -S (заглавная S), так как он позволяет получить сведения как по корневой папке, так и по вложенным в неё директориям:

$ du -S /var/log/



*Вызов команды du с ключами -s и -S*  
  
Нам нужно найти директории, на которые приходится больше всего дискового пространства, поэтому список, который выдаёт du, надо отсортировать, воспользовавшись командой sort:

$ du -S /var/log/ | sort -rn



*Отсортированный список объектов*  
  
Ключ -n указывает команде на то, что нужна числовая сортировка, ключ -r — на обратный порядок сортировки (самое большое число окажется в начале списка). Полученные данные вполне подходят для наших целей.  
  
Для того, чтобы ограничить полученный список первыми десятью записями, воспользуемся [потоковым редактором](https://habrahabr.ru/company/ruvds/blog/327530/) sed, который позволит удалить из полученного списка все строки, начиная с одиннадцатой. Следующий шаг — добавить к каждой полученной строке её номер. Тут также поможет sed, а именно — его команда N:

sed '{11,$D; =}' |

sed 'N; s/\n/ /' |

Приведём полученные данные в порядок, воспользовавшись awk. Передадим awk то, что получилось после обработки данных с помощью sed, применив, как и в других случаях, конвейер, и выведем полученные данные с помощью команды printf:

awk '{printf $1 ":" "\t" $2 "\t" $3 "\n"}'

В начале строки выводится её номер, потом идёт двоеточие и знак табуляции, далее — объём дискового пространства, следом — ещё один знак табуляции и имя папки.   
  
Соберём вместе всё то, о чём мы говорили:

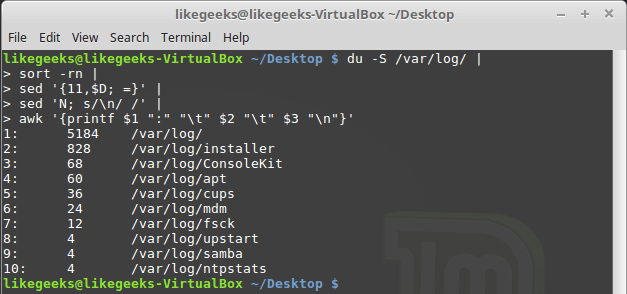
$ du -S /var/log/ |

sort -rn |

sed '{11,$D; =}' |

sed 'N; s/\n/ /' |

awk '{printf $1 ":" "\t" $2 "\t" $3 "\n"}'



*Вывод сведений о дисковом пространстве*  
  
Для того, чтобы повысить эффективность работы скрипта, код которого вы совсем скоро увидите, реализуем возможность получения данных сразу по нескольким директориям. Для этого создадим переменную MY\_DIRECTORIES и внесём в неё список интересующих нас директорий:

MY\_DIRECTORIES="/home /var/log"

Переберём список с помощью цикла for и вызовем вышеописанную последовательность команд для каждого элемента списка. Вот что получилось в результате:

#!/bin/bash

MY\_DIRECTORIES="/home /var/log"

echo "Top Ten Disk Space Usage"

for DIR in $MY\_DIRECTORIES

do

echo "The $DIR Directory:"

du -S $DIR 2>/dev/null |

sort -rn |

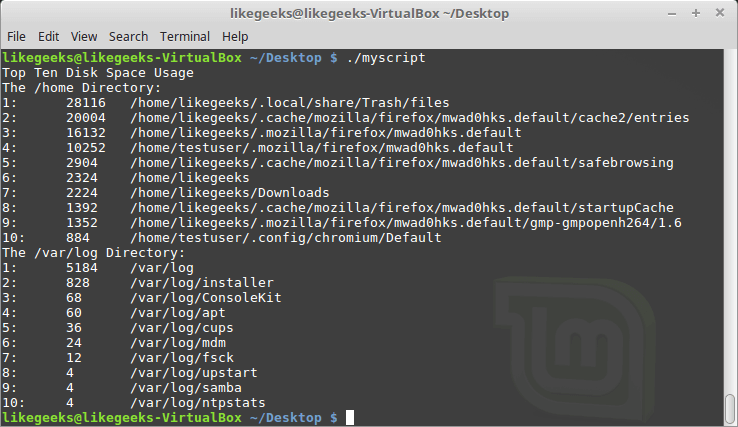
sed '{11,$D; =}' |

sed 'N; s/\n/ /' |

awk '{printf $1 ":" "\t" $2 "\t" $3 "\n"}'

done

exit



*Получение сведений о нескольких директориях*  
  
Как видите, скрипт выводит, в виде удобного списка, сведения о директориях, список которых хранится в MY\_DIRECTORIES.  
  
Команду du в этом скрипте можно вызвать с другими ключами, полученный список объектов вполне можно отфильтровать, в целом — тут открывается широкий простор для самостоятельных экспериментов. В результате, вместо работы со списком папок, можно, например, найти самые большие файлы с расширением .log, или реализовать более сложный алгоритм поиска самых больших (или самых маленьких) файлов и папок.

Итоги

Сегодня мы подробно разобрали пару примеров разработки скриптов. Тут хотелось бы напомнить, что наша главная цель — не в том, чтобы написать скрипт для отправки сообщений с помощью команды write, или сценарий, который помогает в поиске файлов и папок, занимающих много места на диске, а в описании самого процесса разработки. Освоив эти примеры, поэкспериментировав с ними, возможно — дополнив их или полностью переработав, вы научитесь чему-то новому, улучшите свои навыки разработки bash-скриптов.  
  
На сегодня это всё. В следующий раз поговорим об автоматизации работы с интерактивными утилитами с помощью expect.