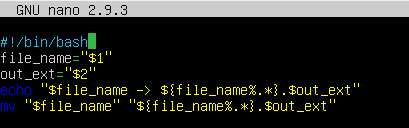
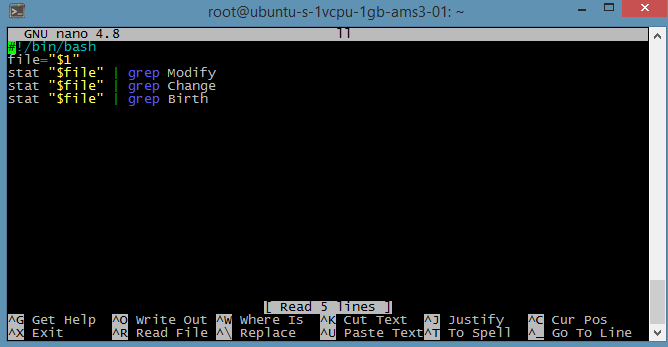
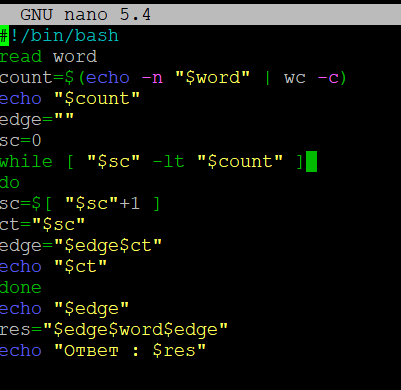
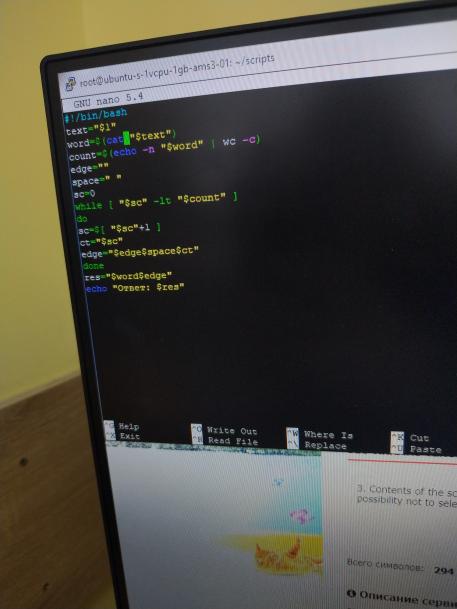
1. Содержание скрипта: замена существующего расширения в имени файла на заданное. Исходное имя файла и новое расширение передаются скрипту в качестве параметров. Основное средство: нестандартное раскрытие переменных. Усложнение: предусмотреть штатную реакцию на отсутствие расширения в исходном имени файла.

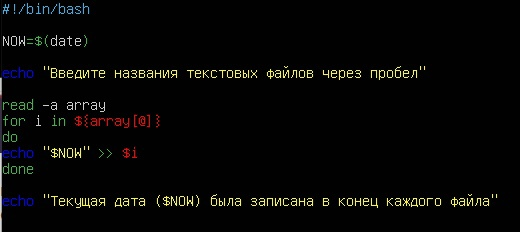


1. Содержание скрипта: выдача сразу трех временных штампов для заданного имени файла. Основные средства: переменные оболочки, gawk, командная подстановка. Усложнение: обработка произвольного числа имен, реакция на ошибки. 
2. Содержание скрипта: выделение из исходной строки подстроки с границами, заданными порядковыми номерами символов в исходной строке. Усложнение: предусмотреть возможность не выделения, а удаления подстроки. Основные средства: команда cut, переменные оболочки.

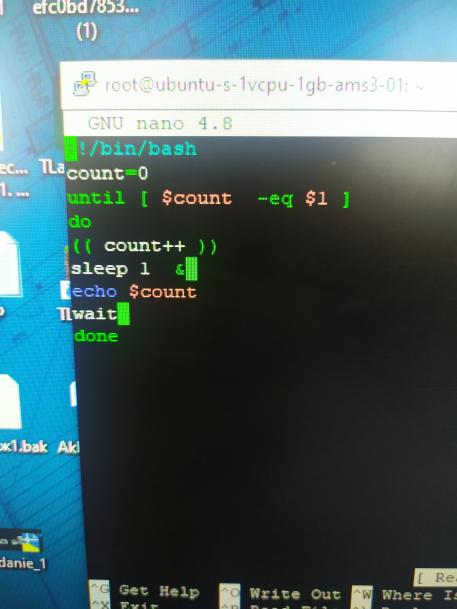




1. Содержание скрипта: добавление в конец каждого текстового файла текущего временного штампа. Имена текстовых файлов передаются скрипту в качестве параметров, их число заранее не известно. Основные средства: структура for, переадресация.



1. Содержание скрипта: организация таймера — периодическая выдача на stdout сообщения о том, сколько времени прошло после запуска таймера (т.е. скрипта) и сколько осталось до конца работы. Параметры таймера передать при запуске скрипта через параметры. Основные средства: структуры while или until, средства для организации счетчика.



1. Содержание скрипта: получение рекурсивного списка файлов, имеющих длинные имена (больше заданного числа), сортировка по полному имени файлов (включающему полное путевое имя, начиная с /) и выдача полученного списка во вьюер и, одновременно, в файл. Параметры скрипта: имя директории, в которой производится рекурсивный поиск, и натуральное число — минимальная длина имени. Основные средства: структура for, команда find, нестандартное раскрытие переменной, использование готового списка файлов, команда tee
2. Разработать сценарий, который выводит на экран количество переданных ему аргументов. Скопировать его в $HOME/bin для дальнейшего использования его в других сценариях.
3. Разработать сценарий, который вызывает предыдущий сценарий дважды: первый раз с объединённым полным списком аргументов, второй раз - со списком всех переданных ему аргументов по отдельности.
4. Разработать сценарий-тестировщик, который будет тестировать ваши сценарии на различные входные данные. Первым аргументом данный сценарий должен принимать имя файла сценария, который необходимо протестировать. Запрограммируйте следующие сценарии тестирования:

с аргументами "1" "2" "3";

с пятью случайными числами (см. переменную $RANDOM);

с аргументами "foo" "bar" "foobar" "foo bar";

с аргументами "foo" "--foo" "--help" "-l".

Протестируйте с помощью разработанного сценария-тестировщика два предыдущих сценария.

1. Перепишите задание так, чтобы исследуемый каталог передавался через аргумент командной строки.

Разработать сценарий, который выводит файлы из текущего каталога в следующем порядке:

1. Каталоги.
2. Обычные файлы.
3. Символьные ссылки.
4. Символьные устройства.
5. Блочные устройства.

Формат вывода:

Каталоги:

drwxr-xr-x 2 root root 560 сен 13 01:34 block/

drwxr-xr-x 2 root root 120 сен 13 01:34 bsg/

drwxr-xr-x 3 root root 60 июн 19 06:41 bus/

drwxr-xr-x 2 root root 3680 сен 13 01:34 char

...

Обычные файлы:

...

Символьные ссылки:

...

Символьные устройства:

...

Блочные устройства:

...

Когда сценарий будет готов, скопируйте его в каталог ~/bin для тестирования. Далее протестируйте сценарий для каталогов /, /dev, /tmp.

#!/bin/bash

cd $1

echo -e "Каталоги:\n$(ls -l |grep ^d)"

echo -e "ОБычные файлы:\n$(ls -l |grep ^-)"

echo -e "Символьные ссылки:\n$(ls -l |grep ^l)"

echo -e "Символьные устройства:\n$(ls -l |grep ^c)"

echo -e "Блочные устройства:\n$(ls -l |grep ^b)"

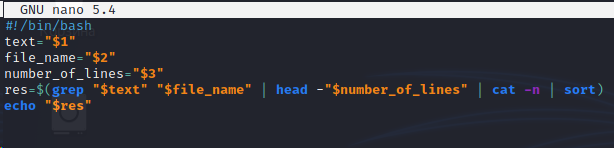
1. Разработать сценарий, который вызывает команду grep и принимает следующие аргументы:

текст, который нужно найти;

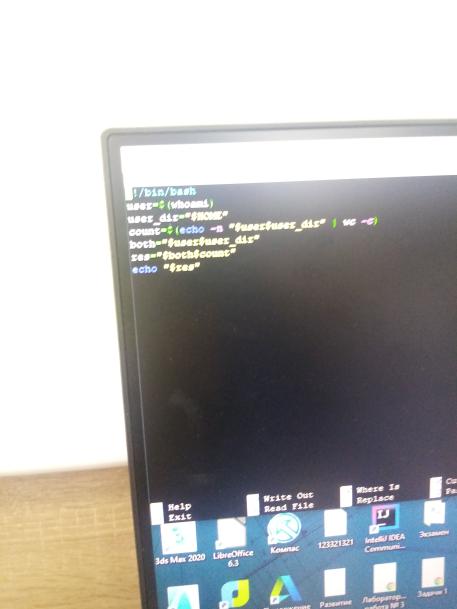
файл, в котором нужно найти этот текст;

максимальное количество строк, которое нужно вывести на экран.

Вывод команды grep отсортировать и пронумеровать.



1. Разработать сценарий, который выводит в одну строку имя пользователя, его домашний каталог, а также количество символов в этих двух переменных. Например: root /root 9. Подсказка: изучите аргументы команды echo, wc, математические вычисления в Bash - $(()).



1. Разработать сценарий для создания полноценной камеры слежения, которая делает снимки в моменты, когда юзер прикасается к мыши.

#!/bin/bash

while true; do

sudo cat /dev/input/mouse0 | read -n1

mpv tv:// -frames 3 -vo jpeg

mv 00000003.jpg date +%F-%H-%M.jpg

rm -f 0000\*.jpg

sleep 10

done

Альтернатива

#!/bin/bash

while true;

do

xinput test 9 | read

scrot ss/$(date +%F-%H-%M-%S).png

done

1. Разработать сценарий для Подключения и настройки внешний монитор из командной строки.

#!/bin/bash

# настройка монитора при подключении через VGA, где $1 - аргумент разрешения монитора (указывается в формате 1920x1080 - пример)

##xrandr --newmode "$1" 148.5 1920 2008 2052 2200 1080 1089 1095 1125 +hsync +vsync

##xrandr --addmode VGA1 $1

##source $HOME/bin/bash\_trace

# дефолтным монитором указан LVDS1 (экран ноутбука)

MONITOR=LVDS1

# функция смены LVDS1 на HDMI (и наоборот)

# $2 - аргумент разрешения экрана ноутбука (формат тот же)

function ActivateHDMI {

echo "Switching to HDMI"

##xrandr --output HDMI1 --mode $1 --dpi 160 --output LVDS1 --off

##xrandr --output HDMI1 --same-as LVDS1

xrandr --output HDMI1 --mode $1

xrandr --output LVDS1 --mode $2

MONITOR=HDMI1

}

function DeactivateHDMI {

echo "Switching to LVDS1"

xrandr --output HDMI1 --off --output LVDS1 --auto

MONITOR=LVDS1

}

# функция проверки подключён ли HDMI и используется

function HDMIActive {

[ $MONITOR = "HDMI1" ]

}

function HDMIConnected {

! xrandr | grep "^HDMI1" | grep disconnected

}

#Функция проверки подключения и активации HDMI-подключения

while true

do

if HDMIConnected

then

ActivateHDMI

fi

if ! HDMIConnected

then

DeactivateHDMI

fi

sleep 1s

done

Alternative

#!/bin/sh

if [ -z "$1" ]; then

exit

fi

if [ $1 == "off" ]; then

xrandr --output VGA-0 --off

xrandr -s 0 else if [ $1 == "on"]; then

xrandr --output LVDS --auto --primary --output VGA-0 --auto --left-of LVDS

xrandr --newmode "1920x1080" 173.00 1920 2048 2248 2576 1080 1083 1088 1120 -hsync +vsync xrandr --addmode VGA-0 1920x1080

xrandr --output VGA-0 --mode 1920x1080

fi

xrandr --dpi 96

альтернатива

#!/bin/bash

# настройка монитора при подключении через VGA, где $1 - аргумент разрешения монитора (указывается в формате 1920x1080 - пример)

##xrandr --newmode "$1" 148.5 1920 2008 2052 2200 1080 1089 1095 1125 +hsync +vsync

##xrandr --addmode VGA1 $1

##source $HOME/bin/bash\_trace

# дефолтным монитором указан LVDS1 (экран ноутбука)

MONITOR=LVDS1

# функция смены LVDS1 на HDMI (и наоборот)

# $2 - аргумент разрешения экрана ноутбука (формат тот же)

function ActivateHDMI {

echo "Switching to HDMI"

##xrandr --output HDMI1 --mode $1 --dpi 160 --output LVDS1 --off

##xrandr --output HDMI1 --same-as LVDS1

xrandr --output HDMI1 --mode $1

xrandr --output LVDS1 --mode $2

MONITOR=HDMI1

}

function DeactivateHDMI {

echo "Switching to LVDS1"

xrandr --output HDMI1 --off --output LVDS1 --auto

MONITOR=LVDS1

}

# функция проверки подключён ли HDMI и используется

function HDMIActive {

[ $MONITOR = "HDMI1" ]

}

function HDMIConnected {

! xrandr | grep "^HDMI1" | grep disconnected

}

#Функция проверки подключения и активации HDMI-подключения

while true

do

if HDMIConnected

then

ActivateHDMI

fi

if ! HDMIConnected

then

DeactivateHDMI

fi

sleep 1s

done

1. Разработать скрипт для получения первых десяти результатов веб поиска.

#!/bin/bash

Q="$@" URL='<https://www.google.de/search?tbs=li:1&q=>'

AGENT="Mozilla/4.0"

stream=$(curl -A "$AGENT" -skLm 10 "${URL}${Q//\ /+}" | grep -oP '/url?q=.+?&amp' | sed 's|/url?q=; s|&amp') echo -e "${stream//%/\x}"

Альтернатива

#!/bin/bash

if [ "$1" = "help" ]; then

#Проверяем если текст help, то выведем помощь

echo "Использование:"

echo " $0 <text>"

echo "text - текст запроса."

echo "Если используется несколько слов, то заключить в кавычки"

exit

fi

#Иначе продолжим программу

#Подготовим запрос

URL="https://go.mail.ru/search?q=$(echo $1 | sed 's/ /%20/g')"

#И параметры HTTP-заголовков для отправки

USERAGENT="Mozilla/5.0 (X11; Linux x86\_64; rv:78.0) Gecko/20100101 Firefox/78.0"

ACCEPT="text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8"

ACCEPTLANG="ru,en-US;q=0.7,en;q=0.3"

ACCEPTENC="gzip, deflate, br"

#Выполняем запрос на сервер, получаем строку JSON с помощью grep и чистим некоторые ненужные символы с помощью sed. В сухом остатке остается просто структура JSON

RESULT=$(curl -sS -L -X "GET" -H "User-Agent: $USERAGENT" -H "Accept: $ACCEPT" -H "Accept-Language: $ACCEPTLANG" --data-urlencode "q=$1" $URL | grep 'go.dataJson = ' | sed 's/go\.dataJson =//' | sed 's/;$//')

#Перебираем массив с полученными с помощью утилиты jq, которая проходит по структуре и "выдергивает" необходимые поля

TITLE=$(echo $RESULT | jq -c -r '.serp.results[] | {number, title, url}')

#Возьмем поле title

TITLE=$(echo $TITLE | jq '.title')

#Немного уберем лишний тэг с помощью sed

TITLE=$(echo "$TITLE" | sed 's/<[\/]\*b>//g')

#Выведем на экран

echo "$TITLE"

1. Твиттер можно использовать не только для выполнения команд, но и для мониторинга машины. Разработать скрипт отправляющий в ленту сообщение с информацией о состоянии машины (имя хоста, uptime, нагрузка, свободная память и нагрузка на CPU).

#!/bin/bash HOST=hostname -s UP=uptime | cut -d" " -f4,5 | cut -d"," -f1 LOAD=uptime | cut -d":" -f5,6 MEM=ps aux | awk '{ sum += $4 }; END { print sum }' CPU=ps aux | awk '{ sum += $3 }; END { print sum }' tweet="Host: ${HOST}, uptime: ${UP}, cpu: ${CPU}%, memory: ${MEM}%, loadavg ${LOAD}" if [ $(echo "${tweet}" | wc -c) -gt 140 ]; then echo "FATAL: The tweet is longer than 140 characters!" exit 1 fi echo $tweet | ttytter -script

если не найдете ttytter