

# ОС UNIX. Практика 1

## Оболочка и скриптовые инструменты

Михаил Пожидаев

15 сентября 2023 г.

# Понятие оболочки UNIX

Функции оболочки:

1. Интерактивная обработка команд.
2. Исполнение скриптов.

Все оболочки условно делятся на поддерживающие стандарт POSIX Shell (bash) и не поддерживающие (csh).

# Стандартные потоки ввода/вывода

- ▶ *stdout* — стандартный вывод, перенаправляется в файл оператором «>»
- ▶ *stdin* — стандартный ввод, перенаправляется из файла оператором «<>»
- ▶ *stderr* — стандартный вывод для ошибок, перенаправляется в файл оператором «2>»

В `bash` весь вывод команды можно перенаправить в файл оператором «&>». Вывод серии команд можно перенаправить одним оператором, взяв их вызов в круглые скобки.

# Примеры вызова команд

- ▶ `make && make install` — вызвать `make install`, только если `make` завершился с кодом ошибки 0;
- ▶ `tar -c foobar/ | gzip > foobar.tar.gz` — упаковать `foobar` в архив и сжать при помощи `gzip`;
- ▶ `find -iname '*.mp3' | wc -l` — подсчитать количество файлов MP3 в текущем каталоге.

# Переменные окружения

Имена переменных окружения начинаются с символа «\$». Команда `export` позволяет сделать переменную видимой для дочерних процессов. Могут быть как пользовательские, так и системные, значение которых учитывается различными командами.

Примеры:

- ▶ `$PATH` — список путей для вызова команд;
- ▶ `export PATH=$PATH:/opt/java/bin` — добавить `/opt/java/bin` в список каталогов для команд;
- ▶ `$LANG` — текущая локаль (`LANG=C date`).

# Особые переменные

- ▶ \$1, \$2, ... — аргументы вызова скрипта;
- ▶ \$@ — все аргументы вызова скрипта;
- ▶ \$? — код завершения предыдущей команды;
- ▶ \$(...) — вызвать команду и сохранить вывод (D=\$(LANG=C date)).

# Подстановки файлов

- ▶ \* — произвольная подстрока (\*.txt)
- ▶ ? — любой символ (???.txt)
- ▶ "... " — исключение подстановок с возможностью использования переменных
- ▶ '...' — исключение подстановок без возможности использования переменных

# Строковые операции

- ▶ `${1%%/*}` — удалить текст в переменной `1` после первого символа `/` включительно;
- ▶ `${1##*/}` — удалить текст в переменной `1` до последнего символа `/` включительно;
- ▶ `${1/.wav/.mp3}` — заменить в переменной `1` подстроку «.wav» на строку «.mp3».



# Цикл «for»

```
for i in *.wav; do
    lame -b 128 $i ${i/.wav/.mp3}
done
```

## Цикл «while»

```
# Enumerate the text of the file line by line
while read l; do
    ...
done < input.txt

# Enumerate the results of the file search
find -iname '*.txt' | while read l; do
    ...
done
```

# Проверка условий

```
# Checking that the file exists
if [ -e file1.txt ]; then
    ...
fi
```

```
# Do some actions, if the file is not corrupted
if shasum -c --status shasum.txt; then
    ...
fi
```

# Проверки утилиты «test»

- ▶ `-r file1` — проверить, что `file1` доступен на чтение;
- ▶ `-d dir1` — проверить, что `dir1` является каталогом;
- ▶ `int1 -eq int2` — проверить `int1` и `int2` на целочисленное равенство;
- ▶ `int1 -gt int2` — проверить, что `int1` строго больше `int2`.

# Функции

```
compress()  
{  
    tar -c "$1"/ | gzip > "$2"  
}
```

```
compress mydir1 mydir1.tar.gz
```

# Обработка сигналов

```
exit_handler()
{
    local rc=$?
    trap - EXIT
    rm -f -- /tmp/socket
    exit $rc
}
trap exit_handler EXIT HUP INT QUIT PIPE TERM
```

# Файловые блокировки

```
(  
    flock -n 9 || exit 1  
    # ... commands executed under lock ...  
) 9>/var/lock/mylockfile
```

# Порядок исполнения скриптов

Скрипт должен начинаться с так называемого шебанга, который задаёт имя интерпретатора.

Пример: `#!/bin/bash -e`

Существует ряд скриптов, вызываемых автоматически в ходе различных операций, таких как инициализация пользовательских сессий, создание новых оболочек и пр.

Пример: `~/.bashrc`



# Популярные утилиты для использования в скриптах

- ▶ `echo` — вывести строку на экран;
- ▶ `cat` — перенаправить стандартный ввод на стандартный вывод;
- ▶ `wc` — подсчёт параметров текста;
- ▶ `grep` — обработать текст с использованием регулярных выражений;
- ▶ `xargs` — вызвать указанную утилиту с параметрами, полученными на стандартном вводе, включая параллельный режим запуска;
- ▶ `sed` — отредактировать текст на основе заданных правил в потоковом режиме;
- ▶ `sox` — произвести трансформацию аудиоданных.

# Параллельное кодирование данных

```
find -iname '*.wav' | time xargs -P 1 -n 1 flac  
21.64user 1.76system 0:23.68elapsed 98%CPU
```

```
find -iname '*.wav' | time xargs -P 8 -n 1 flac  
22.41user 2.21system 0:11.22elapsed 219%CPU
```

```
find -iname '*.wav' | time xargs -P 16 -n 1 flac  
22.68user 1.85system 0:07.86elapsed 312%CPU
```

# Потоковые редакторы текста

`sed` — потоковый редактор текста с декларативным заданием преобразований. Пример команды для удаления текста до первого двоеточия:

```
sed -e s/'^[^:]*:\(.*\)$/'\1'/
```

`awk` — потоковый редактор текста с императивным заданием преобразований. Пример команды для вывода всех прав доступа в текущем каталоге:

```
ls -l | awk '{print($1);}'
```

# Спасибо за внимание!

Всё о курсе: <https://marigostra.ru/materials/unix.html>

E-mail: [msp@luwrain.org](mailto:msp@luwrain.org)

Канал в Телеграм: <https://t.me/MarigostraRu>