

Операционни системи

ФМИ

СИ 2017

Shell



Shell

Предоставя:

- ▶ Команден интерфейс
- ▶ Начин за управление на процесите
- ▶ Пускане и спиране на приложения
- ▶ Автоматизиране на задачи

Shell-ове за Unix/Linux

- ▶ Thompson Shell (**sh**) - Ken Thompson, 1971 - прост и примитивен
- ▶ Bourne Shell (**sh**) - Stephen Bourne, 1977 - прост и по-малко примитивен
- ▶ C Shell (**csh**) - Bill Joy, 1978 - синтаксис, приличащ на C
- ▶ Korn Shell (**ksh**) - David Korn, 1983
- ▶ Enhanced C Shell (**tcsh**) - Ken Greer, 1983
- ▶ Bourne Again Shell (**bash**) - Brian Fox, 1989 - доста по-умен от sh, използва се по подразбиране почти навсякъде
- ▶ Debian Almquist shell (**dash**) - Herbert Xu, 1997
- ▶ Z Shell (**zsh**) - Paul Falstad, 1990 - добре разширяем shell с много плъгини

Default shell

- ▶ Различен за всеки потребител
- ▶ Пише го в `/etc/passwd`

Смяна на shell

- ▶ `$ dash` - за да се смени текущият shell с друг, може да се напише името му в терминала. Shell-ът е executable file, текущият шел просто го пуска.
- ▶ За постоянно се сменя с `$ chsh`
- ▶ В `/etc/shells` има списък с позволените shell-ове

Ще говорим за bash.

Login shell

- ▶ първият shell, който се стартира, когато се логнем в системата, се нарича Login shell. Той прочита profile-файлове (`~/.profile`, `/etc/profile`), в които стоят някакви настройки, които трябва да се изпълнят веднъж на логин.
- ▶ всички shell-ове прочитат `~/.bashrc`, в който има настройки за всеки shell.

Alias

▶ alias

```
$ alias 'c=clear'
```

▶ unalias

```
$ alias 'foo=command'  
$ unalias foo
```

Shell scripts

Shell scripts

- ▶ Скриптът е текстов файл, в който има изредени команди, които се изпълняват една след друга

```
echo "those are all running processes:"  
ps -e  
echo "those are the files in the current dir:"  
ls
```

- ▶ Изпълнение - подаваме го като аргумент на shell: `bash foo.sh`
- ▶ Коментари - след `#`

```
echo "foo"  # this is a comment
```

Директно изпълнение

- ▶ Shebang - указва с кой shell да се изпълни файла - `#!/bin/bash`
 - ▶ задължително е първият ред от скрипта
- ▶ Изпълнение на скрипт: с пълен или относителен път
 - ▶ `$ /home/human/foo.sh`, `$./foo.sh`
 - ▶ трябва да имаме execute permission

Променливи

Shell variables

- ▶ Виждат се само в текущия shell
- ▶ `foo=blablabla` - дефинира променлива
 - ▶ няма спейсове около =
- ▶ `foo="foo bar baz"` - променлива, съдържаща спейсове
- ▶ `$ set` - показва и манипулира променливите
- ▶ `$ unset foo` - унищожава променливата `foo`

Променливи

- ▶ Стойността може да се подаде и с read

```
$ read foo bar baz
one two three # Enter
$ echo "${foo}"
one
$ echo "${bar}"
two
$ echo "${baz}"
three
```

Променливи

```
$ read foo bar  
one two three # Enter  
$ echo "${foo}"  
one  
$ echo "${bar}"  
two three
```

Променливи

Използване на променливи

```
$ answer=42  
$ echo $answer  
42  
$ echo "$answer"  
42  
$ echo "${answer}"  
42
```


Променливи

Каква е разликата?

- ▶ Първият метод ще разшири променливата до различни аргументи

```
$ touch 'foo bar baz'
$ filename="foo bar baz"
$ cat $filename
cat: foo: No such file or directory
cat: bar: No such file or directory
cat: baz: No such file or directory
```

Променливи

- ▶ вторият метод ще фейлне, ако има променлива с префикс-име

```
$ answer=42  
$ an=gel  
$ echo "$answer"  
gelswer
```

Променливи

▶ третият метод е най-сигурен

```
$ answer=42  
$ echo "${answer}"  
42
```

Променливи

Environment variables

- ▶ Глобални са
- ▶ `$ env` - показва и манипулира променливите
- ▶ `$ env -u, $ unset` - унищожава променливи
- ▶ `export FOO="foo"` - прави променливата FOO environment variable
 - ▶ Това се запазва само за текущия shell и децата му
 - ▶ Ако искаме да е наистина глобално, трябва да напишем `export FOO="foo"` в `~/.bash_profile`, `~/.profile` или `/etc/profile`

Environment variables

- ▶ `$PATH` - множество от директориите, в които се намират executable файлове. Shell-ът търси в тях.
- ▶ `$PWD` - показва текущата директория и е еквивалентна на изхода командата `pwd`
- ▶ `$HOME` - показва пътя на home директорията
- ▶ `$USER` - показва username на текущия потребител
- ▶ `$SHELL` - пътят до login shell

and many more

Параметри на shell script

- ▶ За да пишем shell скриптове, трябва да можем да обработваме подадените аргументи

```
$ ./foo.sh bar baz qux
```

- ▶ \$0, \$1, \$2...
 - ▶ \$0 - винаги е името на изпълняваната програма - в случая ./foo.sh
 - ▶ \$1 - първият аргумент - bar
 - ▶ ...

Параметри на shell script

```
$ ./foo.sh bar baz qux
```

- ▶ \$# - брой аргументи без името
 - ▶ 3 в горния пример
- ▶ \$@- списък от всички аргументи
 - ▶ bar baz qux в горния пример
- ▶ \$? - exit status на последната изпълнена команда
 - ▶ 0, ако всичко е наред
 - ▶ 1-255 - код за грешка иначе
 - ▶ exit [n]

Математика в shell

► `$((1+1))`

```
$ foo=1
$ echo $((1 + $foo))
2
```

► `$(1+1)` - deprecated

► `expr 1+1` - old 'n' slow

```
$ foo=1
$ expr 1 + $foo
2
```


Условия в shell

- ▶ `test` - взима условие, което да проверява
 - ▶ връща 0, ако е изпълнено
 - ▶ и 1, ако не е
 - ▶ `test condition`
- ▶ `-eq, -ne, -gt, -lt, -le, -ge`
- ▶ `test condition && do_stuff` - изпълнява `do_stuff`, ако условието е вярно
- ▶ `test condition || do_stuff` - изпълнява `do_stuff`, ако условието **не** е вярно
- ▶ Може и двете заедно:
`test $((6*9)) -eq 42 && echo "Yep" || echo "Wrong"`
- ▶ `[[4 -eq 5]]` е еквивалентен запис за `test 4 -eq 5`

Chain commands

- ▶ Може да се използва `exit` кодът на предната команда като условие.

```
$ grep "42" life.txt && echo "It is the meaning of life"  
$ grep "42" life.txt || echo "Douglas Adams lied to me"
```

```
$ check_for_viruses && clean_system || party_hard
```

Chain commands

- ▶ Може да ползваме && ако държим всички команди по веригата да се изпълнят

```
$ get_data && filter_data && send_data
```

- ▶ Ако exit кодът не ни интересува просто слагаме ;
- ▶ \$ foo; bar
- ▶ по подразбиране новите редове се интерпретират като ;
- ▶ като всичко в Линукс, това може да се смени

If

```
if condition
then
  do_stuff
fi
```

```
if condition; then
  do_stuff
fi
```

```
if condition
then
  do_stuff
else
  other_stuff
fi
```

```
if condition; then
  do_stuff
else
  other_stuff
fi
```

If

```
if condition1
then
  do_stuff
elif condition2
then
  other_stuff
fi
```

```
if condition1; then
  do_stuff
elif condition2; then
  other_stuff
fi
```

Примери

```
#!/bin/bash

if grep "Velin" /etc/passwd; then
    echo "Velin is here"
else
    echo "No Velin in sight"
fi
```

Примери

```
#!/bin/bash

if [[ $# -eq 5 ]]; then
    echo "We have 5 parameters"
fi
```

Примери

```
#!/bin/bash

if [[ "${1}" == "--help" ]]; then
    show_help_page
fi
```


Case

```
case foo in
  bar)
    echo "We have bar"
    ;;
  baz)
    ...
  *)
    echo "else"
    ;;
esac
```

Case

```
case "${1}" in
    start)
        echo "Starting"
        run_start ;;
    stop)
        echo "Stopping"
        run_stop ;;
    restart)
        echo "restarting"
        run_stop && run_start ;;
    *)
        echo "Invalid option" ;;
esac
```

Loops



For

```
for item in foo bar baz
do
    echo "${item}"
done
```

```
for item in foo bar baz; do
    echo "${item}"
done
```

- ▶ Може ръчно да се изброят нещата
`for i in 1 2 3 4 5; do echo "i is ${i}"; done`
- ▶ Може да се подава range
`for i in {1..10}; do echo "i is ${i}"; done`
- ▶ Може да се подава range със стъпка
`for i in {1..10..2}; do echo "i is ${i}"; done`
- ▶ Може да итерира всичко разделено с нещата в `$IFS` (default: space, tab)
`for file in *.txt; do cat "${file}"; done`

For

```
#!/bin/bash

i=0
for parameter in "${@}"; do
    echo "${i}: ${parameter}"
    i=$((i + 1))
done
```

For

- Има break и continue

```
#!/bin/bash

i=0
for parameter in "${@}"; do
    if [[ $i -ge 5 ]]; then
        break
    fi
    echo "${i}: ${parameter}"
    i=$((i + 1))
done
```

While

```
while command  
do  
  do_stuff  
done
```

```
while command; do  
  do_stuff  
done
```

While

- ▶ За условие се подава команда, като се гледа exit status-ът ѝ
- ▶ Може да се подаде test:

```
#!/bin/bash

touch foo.txt
i=0
while [[ $(wc -l < foo.txt) -le 10 ]]; do
    echo "${i}: spam" >> foo.txt
    i=$((i + 1))
done
```


Функции

```
function_name () {  
    list of commands  
}
```

```
function function_name {  
    list of commands  
}
```

- ▶ Функциите си имат собствени аргументи, които се достъпват с \$0, \$1...т.н

Функции

foo.sh

```
#!/bin/bash

function say_hello {
    echo "Hello, ${1}"
}

if [[ ${1} == "--hello" ]]; then
    say_hello "${2}"
fi
```

\$./foo.sh --hello Peasant

Функции

Променливи

- ▶ Променливите във функциите са глобални: ако обявим променлива във функция, ще се вижда и отвън **foo.sh**

```
#!/bin/bash

function foo {
    i=42
}

i=20
foo
echo "${i}" # outputs 42
```

► Можем да ги направим локални:

```
#!/bin/bash

function foo {
    local i=42
}

i=20
foo
echo "${i}" # outputs 20
```