

Operacijski sistemi



Funkcije

Vaje

Vsebina

- Izvajanje v podlupini
- Ukazni bloki
- Funkcije
- Primeri

Izvajanje v podlupini

- Podlupina (subshell)

- podlupina je podproces trenutne lupine
- nudi avtomatsko dedovanje
 - spremenljivke, funkcije, odprte datoteke
- ne deduje
 - rokovalniki signalov

Izvajanje v podlupini

- **Osnovni zagon podlupine**
 - (*ukaz*)

```
( hostname )  
  
( ( ( ( echo -n $BASH_SUBSHELL ) ); echo $BASH_SUBSHELL ) )  
  
( echo Zivjo; exit 42; echo svet. )  
  
( echo ZACETEK; ls -alp; echo KONEC )  
  
pwd; ( cd /; pwd ); pwd
```


Izvajanje v podlupini

- **Zagon podlupine s substitucijo**

- `$ (ukaz)`
- ``ukaz``

```
$( hostname )
```

```
h=$(hostname)
```

```
dirname $(which less)
```

```
u=$(whoami); h=$(hostname); echo Dobrodošel $u na $h
```

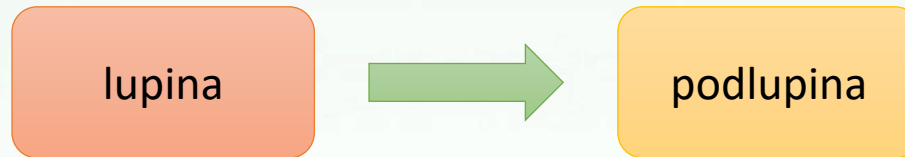
```
echo Dobrodošel $(whoami) na $(hostname)
```

```
data=$(cat /etc/passwd)
```

```
echo $(echo $(echo $(echo $(echo $(echo $RANDOM))))))
```

Izvajanje v podlupini

- Dedovanje spremenljivk



```
# Lupina
x=42; echo $x

# Podlupina
( echo $x; x=1; echo $x )

# Spet lupina
echo $x
```

Ukazni bloki

- Ukazni bloki

```
{ ukazi; }
```

... združevanje ukazov v blok

Pozor: obvezen presledek za { in ; pred }.

Združevanje pogojev

```
true && { false || true; }
```

Združevanje ukazov

```
mkdir test && { cd test; touch dat.txt dat.jpg; }
```

```
[[ $(whoami) == root ]] && { echo "I'm the boss."; cat /etc/shadow; }
```

Ukazni bloki

- Ponovitev (le kot referenca)

<code>{ pwd; ls }</code>	... združevanje ukazov v blok
--------------------------	-------------------------------

<code>{a,b,c}</code>	... posebna razširitev
----------------------	------------------------

<code>(pwd)</code>	... podlupina
----------------------	---------------

<code>((a=1+2))</code>	... aritmetika
--------------------------	----------------

<code>[a = b]</code>	... pogojni izrazi
------------------------	--------------------

<code>[[-d /bin]]</code>	... razširjeni pogojni izrazi
----------------------------	-------------------------------

Funkcije

- Definicija funkcij

```
function ime { ukazi; }  
ime() { ukazi; }
```

... bolj bash-ish
... bolj prenosljivo

Pozor: telo funkcije je blok, veljajo enaka sintaktična opozorila!

```
function f { echo F; }
```

```
function f {  
    echo F  
}
```

```
f() { echo F; }
```

```
f() {  
    echo F  
}
```

Vsi prikazani primeri
so enakovredni.



Funkcije

- Uporaba funkcij

Funkcije "kličemo" enako kot ostale ukaze.

```
function f { echo F; }  
f  
  
function do_big_thing {  
    cat /etc/passwd | cut -d: -f7 | sort | uniq -c  
}  
do_big_thing  
  
klici=f  
$klici
```

Funkcije

- Dedovanje funkcij

Funkcija definirana v lupini

```
function f { echo F; }
```

```
f
```

```
( f )
```

Funkcija definirana v podlupini

```
(
```

```
    function g { echo G; }
```

```
    g
```

```
)
```

```
g
```

Funkcije


- Argumenti funkcije

Enako kot argumenti skripte.

Vgrajene spremenljivke: \$1, \$2, ...

```
function fullname {  
    local name=${1:-Janez}  
    local surname=${2:-Novak}  
    echo $name $surname  
}
```

```
function print_my_args {  
    while [[ -n $1 ]]; do  
        echo $1  
        shift  
    done  
}
```



Koliko argumentov podpirata
ti dve skripti?

Funkcije

- **Rezultat funkcije** `return status`
- *lahko* uporabimo izhodni status

```
function vsota_fail {  
    local vsota=0  
    while [[ -n $1 ]]; do  
        (( vsota += $1 ))  
        shift  
    done  
    return $vsota  
}
```

```
function fak_fail {  
    if (( $1 <= 1 )); then  
        return 1  
    else  
        fak_fail $(( $1 - 1 ))  
        return $(( $1 * $? ))  
    fi  
}
```

FAIL

- Kako bi vrnili niz kot rezultat?
- Zakaj je vračanje rezultata preko izhodnega statusa funkcije napačen pristop?

Funkcije

- Rezultat funkcije
 - raje uporabimo standardni izhod

- `return` nadomestimo z `echo`
- rezultat ujamemo preko izvajanja v podlupini

```
function vsota {  
    local vsota=0  
    while [[ -n $1 ]]; do  
        (( vsota += $1 ))  
        shift  
    done  
    echo $vsota  
    return 0  
}
```

```
function fak {  
    if (( $1 <= 1 )); then  
        echo 1  
    else  
        local tmp=$((fak $(( $1 - 1 )) )  
        echo $(( $1 * $tmp ))  
    fi  
    return 0  
}
```

- izhodni status pa uporabimo za sporočanje uspešnosti izvedbe funkcije

Primeri

- Ogrodje skripte
 - če skripta podpira več akcij

```
function run_update { ... }  
function get_upgrade { ... }  
function print_help { ... }  
  
action=$1  
shift  
case $action in  
    update)      run_update $* ;;  
    upgrade)     get_upgrade $* ;;  
    list_home)   ls -R ~ ;;  
    version)     echo Version 3.14  
    help)        print_help  
esac
```

```
action=$1  
shift  
do_$action $*
```

Primeri

- "*Funkcijsko*" programiranje

```
function map {
    local fun=$1
    shift
    while [[ -n $1 ]]; do
        $fun $1
        shift
    done
}

function to_upper {
    tr '[a-z]' '[A-Z]' <<< $1
}

function inc {
    echo $(( $1 + 1 ))
}

map to_upper miha maja meta
map inc 1 2 3 42 100
```