

```
[user@centos8 ~]$ sudo nano /var/users
[sudo] password for user:
[user@centos8 ~]$ cut -d' ' -f1 /var/users
user30
user31
user32
user33
user34
user35
[user@centos8 ~]$
```

В прошлый раз мы остановились на том, что создали файл и в скрипте добавили возможность брать данные о пользователе и группе из этого файла. Теперь же попробуем в файле указать несколько пользователей - `sudo nano /var/users` и добавить их разом. Выполнив ту же команду `cut -d' ' -f1 /var/users` - мы увидим весь список пользователей. Если передать команде `useradd` такой список пользователей, она этого не поймёт – `useradd` может добавлять только по одному пользователю за раз. А значит нам нужно будет для каждой строки запускать отдельный `useradd`.

```
[user@centos8 ~]$ tail -n +30 myscript
groupadd $group

# Sudoers
if [ "$group" = it ] || [ "$group" = security ]
then
    if ! grep "%$group" /etc/sudoers
    then
        cp /etc/sudoers{,.bkp}
        echo '%'$group' ALL=(ALL) ALL' >> /etc/sudoers
    fi
    shell=/bin/bash
elif [ "$user" = admin ]
then
    if ! grep "$user" /etc/sudoers
    then
        cp /etc/sudoers{,.bkp}
        echo $user' ALL=(ALL) ALL' >> /etc/sudoers
    fi
    shell=/bin/bash
fi
mkdir -v /home/$group
useradd $user -g $group -b /home/$group -s $shell
[user@centos8 ~]$
```

И так, задача у нас такая – для каждой строчки в файле /var/users создавать группу, проверять sudoers и создавать пользователя. То есть всё что ниже 30 строчки — tail -n +30 myscript.

Если речь про повторное запускание одной и той же команды – то речь обычно о циклах. Есть две стандартные команды для работы с циклами – for и while. for обычно связан со списком, а while – с условием, хотя нередко можно использовать и ту, и другую. Давайте начнём с for. Синтаксис такой:

```
for переменная in список значений
do команды
done
```

При запуске команды переменная получит первое значение из списка значений, потом выполнятся все команды, а после команд переменная получит второе значение из списка значений и опять выполнятся все команды. И так будет повторяться до тех пор, пока не закончатся все значения в списке, после чего цикл завершится. Каждое повторение называется итерацией.

```
[user@centos8 ~]$ nano for
[user@centos8 ~]$ cat for
for number in 1 two "line #3"
do echo This is $number
done
[user@centos8 ~]$ chmod +x for
[user@centos8 ~]$ ./for
This is 1
This is two
This is line #3
[user@centos8 ~]$
```

Давайте посмотрим пример — nano for; for number in 1 two “line № 3”; do echo This is \$number; done; chmod +x for; cat for; ./for . Как видите, сначала переменная number получила первое значение – 1, выполнялась команда echo. Потом переменная number взяла второе значение – two. Ну и так далее. Вроде ничего сложного.

```
[user@centos8 ~]$ cat for
for line in $(cat /var/users)
do echo In this line: $line
done
[user@centos8 ~]$ ./for
In this line: user30
In this line: it
In this line: user31
In this line: users
In this line: user32
In this line: it
In this line: user33
In this line: group33
In this line: user34
In this line: it
In this line: user35
In this line: users
[user@centos8 ~]$
```

Список значений можно задать по разному, например, взять его из вывода команды - `nanop for; for line in $(cat /var/users); do echo In this line: $line; done; cat for; sudo ./for`. Но вместо того, чтобы увидеть в виде значения каждую строчку, мы видим пользователя и группу на разных строчках. То есть цикл сначала присвоил переменной `line` в виде значения имя первого юзера, а после итерации значение переменной стало имя группы. И так с каждой строчкой. То есть, вместо того, чтобы разделять значения построчно, значения разделялись по пробелам.

Помните, мы в команде `cut` использовали опцию `-d` – разделитель - `sudo cut -d' ' -f1 /var/users`? И мы этой опцией указали, что разделителем является пробел. `bash`, чтобы взять список значений, тоже использует разделитель – сначала он попытается разделить значения по пробелу, потом по табуляции и только потом по переводу строки. А чтобы `bash` в качестве разделителя использовал сразу перевод строки, нам нужно об этом ему сказать. Для этого есть переменная `IFS` – внутренний разделитель полей.

```
IFS=$'\n'
IFS=$'\t'
IFS=:
oldIFS=$IFS
IFS=$oldIFS
```

Чтобы указать, что мы хотим в качестве разделителя использовать перевод строки, даём переменной такое значение - `IFS=$'\n'`. `\n` – это newline. Если захотим знак табуляции меняем `n` на `t` - `IFS=$'\t'`. Если брать, например, `/etc/passwd`, то там разделителем выступает двоеточие, тогда можно указать так - `IFS=:`. Но с этой переменной нужно быть осторожным – другие команды в скрипте также могут использовать эту переменную, а значит то что у вас работало с пробелами, может начать работать с новыми строками. И чтобы не пришлось переделывать пол скрипта, мы можем поменять эту переменную временно, а потом вернуть старое значение. Но для этого нужно старое значение предварительно сохранить - `oldIFS=$IFS`. После выполнения нужных команд можем восстановить старое значение `IFS=$oldIFS`.

```
[user@centos8 ~]$ nano for
[user@centos8 ~]$ cat for
IFS=$'\n'
for line in $(cat /var/users)
do echo This is line: $line
done
[user@centos8 ~]$ ./for
This is line: user30 it
This is line: user31 users
This is line: user32 it
This is line: user33 group33
This is line: user34 it
This is line: user35 users
[user@centos8 ~]$
```

Но нам сейчас нужен newline - `IFS=$'\n'`. Попробуем запустить скрипт - `sudo ./for`. Теперь всё окей – при каждой итерации переменная будет получать в качестве значения целую строку.

```

[user@centos8 ~]$ nano for
[user@centos8 ~]$ cat for
oldIFS=$IFS
IFS=$'\n'
for line in $(cat /var/users)
do
    user=$(echo $line | cut -d' ' -f1)
    group=$(echo $line | cut -d' ' -f2)
    echo Username: $user    Group: $group
done
IFS=$oldIFS
[user@centos8 ~]$ ./for
Username: user30 Group: it
Username: user31 Group: users
Username: user32 Group: it
Username: user33 Group: group33
Username: user34 Group: it
Username: user35 Group: users
[user@centos8 ~]$ █

```

Дальше нужно просто из этой переменной достать имя пользователя и группы, допустим, с помощью того же cut. А чтобы передать команде cut значение переменной, можно использовать пайп - echo \$line | cut -d' ' -f1. В итоге мы достанем из строки имя пользователя. И чтобы это имя стало переменной, напишем так - user=\$(echo \$line | cut -d' ' -f1). Тоже самое с группой - group=\$(echo \$line | cut -d' ' -f2). Последний штрих - echo Username: \$user Group: \$group; cat for; sudo ./for. Как видите, всё сработало как надо. Теперь попытаемся сделать тоже самое с нашим скриптом.

```

# Check arguments
if [ ! -z $2 ]
then
    user=$1
    group=$2
    echo Username: $user    Group: $group
elif [ -f $file ]
then
IFS=$'\n'
for line in $(cat $file)
do
    user=$(echo $line | cut -d' ' -f1)
    group=$(echo $line | cut -d' ' -f2)
    echo Username: $user    Group: $group
done
IFS=$oldIFS

```

Предварительно сохраним значение переменной IFS - oldIFS=IFS. Тут у нас уже есть строки назначения переменных из файла, но это нам не подходит, потому что нам нужно брать переменные в цикле, поэтому эти строки убираем. Попробуем вписать сюда наш цикл - IFS=\$'\n'; for line in \$(cat \$file); do user=\$(echo \$line | cut -d' ' -f1); group=\$(echo line | cut -d' ' -f2); echo Username: \$user Group: \$group; done; IFS=\$oldIFS.

```

[user@centos8 ~]$ sudo ./myscript
[sudo] password for user:
Username: user30 Group: it
Username: user31 Group: users
Username: user32 Group: it
Username: user33 Group: group33
Username: user34 Group: it
Username: user35 Group: users
groupadd: group 'users' already exists
mkdir: cannot create directory '/home/users': File exists
[user@centos8 ~]$ tail -5 /etc/passwd
user16:x:1124:1009:./home/group16/user16:/sbin/nologin
user17:x:1125:1010:./home/group17/user17:/sbin/nologin
user18:x:1126:1011:./home/group18/user18:/sbin/nologin
user22:x:1127:1004:./home/it/user22:/bin/bash
user35:x:1128:100:./home/users/user35:/sbin/nologin
[user@centos8 ~]$

```

Запустим и проверим - sudo ./myscript; tail -5 /etc/passwd. У нас там было несколько пользователей, а создался только последний. Подумайте, почему так случилось?

Обратите внимание на наш цикл – переменные получают свои значения, выполняется команда `echo`, а после итерации всё происходит заново, пока не дойдёт до последнего значения. И только после этого начинают выполняться все остальные команды – группы, `sudoers` и т.д. Нам же нужно, чтобы с каждой итерацией выполнялись все эти команды.

Что мне мешает поставить `done` в конце скрипта? Мы сейчас находимся в условии – я не могу просто посреди `for` закончить условие `if` и продолжить выполнять команды `for`. Команда началась внутри условия – там же она должна закончиться. Есть вариант скопировать все оставшиеся команды сюда. Но это плохой вариант – это увеличит размер скрипта, в дальнейшем придётся редактировать команды и внутри цикла, и отдельно.

```
create_user() {
IFS=oldIFS
groupadd $group

# Sudoers
if [ "$group" = it ] || [ "$group" = security ]
then
    if ! grep "%$group" /etc/sudoers
    then
        cp /etc/sudoers{,.bkp}
        echo '%$group' ALL=(ALL) ALL' >> /etc/sudoers
    fi
    shell=/bin/bash
elif [ "$user" = admin ]
```

Вот у нас есть куча команд и я не хочу, чтобы они повторялись в скрипте в нескольких местах. Чтобы решить эту проблему, я могу объединить все эти команды под одним названием. Для этого я пишу название, допустим, `create_user()` - ставлю после названия скобки, а потом внутри фигурных скобок указываю все нужные команды - `create_user() { groupadd ... }`. Это называется функцией. И в дальнейшем, когда я захочу запустить все эти команды, я просто запущу команду `create_user`. Но функция должна быть задана до того, как к ней обращаются, поэтому переместим нашу функцию наверх, скажем, после переменных. Но тут ещё один нюанс – `IFS` возвращает старое значение - `IFS=$oldIFS` после выполнения цикла, а значит после выполнения всех команд в функции. А так как все наши команды там, то лучше перенести эту команду - `IFS=$oldIFS` - в начало функции.


```

then
    user=$1
    group=$2
    echo Username: $user    Group: $group
    create_user
elif [ -f $file ]
then
IFS=$'\n'
for line in $(cat $file)
do
    user=$(echo $line | cut -d' ' -f1)
    group=$(echo $line | cut -d' ' -f2)
    echo Username: $user    Group: $group
    create_user
done
IFS=$oldIFS
else
    echo Welcome!

```

А теперь пропишем её в наших условиях – просто написав create_user в командах каждого из условий.

Хорошо, давайте пройдемся по скрипту. Вначале мы проверяем на root права. Задаём переменные. Создаём функцию – create_user – тут у нас все нужные команды для создания группы и пользователя. А в конце у нас проверка, как мы запускали программу – с параметрами, с файлом или интерактивно – в зависимости от этого назначаются переменные и запускается функция.

```

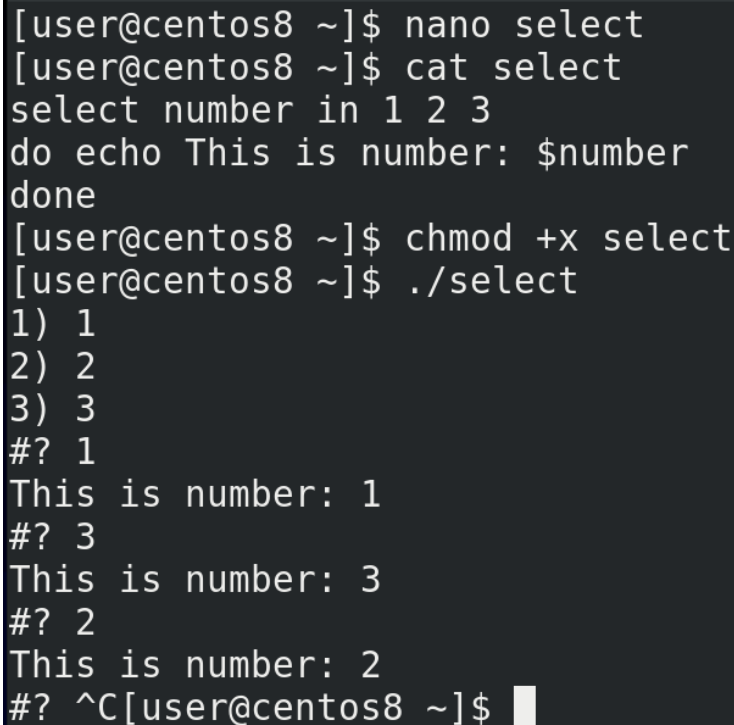
mkdir: cannot create directory '/home/it': File exists
Username: user35 Group: users
groupadd: group 'users' already exists
mkdir: cannot create directory '/home/users': File exists
useradd: user 'user35' already exists
[user@centos8 ~]$ tail -5 /etc/passwd
user30:x:1129:1004::/home/it/user30:/bin/bash
user31:x:1130:100::/home/users/user31:/bin/bash
user32:x:1131:1004::/home/it/user32:/bin/bash
user33:x:1132:1012::/home/gr/user33:/bin/bash
user34:x:1133:1004::/home/it/user34:/bin/bash
[user@centos8 ~]$ █

```

Окей, давайте протестируем - sudo ./myscript; tail -5 /etc/passwd. Как видите, все пользователи создались, всё работает.

Теперь немного проработаем наше интерактивное меню, то есть опцию else. Сейчас, при запуске скрипта, в этом режиме оно запросит юзернейм, пароль, создаст пользователя и закроется. Я бы хотел, чтобы после создания пользователя наш скрипт не завершался, а предлагал заново создать пользователя и всякие другие менюшки. Для этого мне понадобятся две команды. Первая будет показывать меню – это команда select. Синтаксис чем-то похож на for:

```
select переменная in список значений
do команды
done
```



```
[user@centos8 ~]$ nano select
[user@centos8 ~]$ cat select
select number in 1 2 3
do echo This is number: $number
done
[user@centos8 ~]$ chmod +x select
[user@centos8 ~]$ ./select
1) 1
2) 2
3) 3
#? 1
This is number: 1
#? 3
This is number: 3
#? 2
This is number: 2
#? ^C[user@centos8 ~]$
```

Например - nano select; select number in 1 2 3; do echo This is number: \$number; done; chmod+x select; ./select. select показал нам меню, где с помощью цифр мы можем выбрать какое-то из значений и переменная получит это значение. Дальше выполнится команда и после неё опять появится меню.

Теперь мне нужна команда, которая будет запускать какие-то команды в зависимости от значения переменной. Речь про команду case. Синтаксис такой:

```
case $переменная in
    значение 1) команды;;
    значение 2) команды ;;
    *) команды, если значения нет в списке ;;
esac
```

```
[user@centos8 ~]$ nano case
[user@centos8 ~]$ cat case
number=one
case $number in
    one) echo 1;;
    two) echo 2;;
    *) echo something wrong;;
esac
[user@centos8 ~]$ chmod +x case
[user@centos8 ~]$ ./case
1
[user@centos8 ~]$
```

Например, nano case, number=one; case \$number in one) echo 1;; two) echo 2;; *) echo something wrong ;; esac ; chmod +x case; ./case. Как видите, значение переменной было one. Оно подошло под первую опцию, в следствии чего сработала первая команда.

```
[user@centos8 ~]$ cat case
select number in 1 2 3
do
    case $number in
        1) echo One;;
        2) echo Two;;
        3) echo Three ;;
        *) echo something wrong;;
    esac
done
[user@centos8 ~]$ ./case
1) 1
2) 2
3) 3
#? 1
One
#? 3
Three
#? 4
something wrong
#? █
```

Теперь объединим select и case. Например - select number in 1 2 3; do case \$number in 1) echo One;; 2) echo Two;; 3) echo Three;; *) echo something wrong ;; esac ; done; ./case. Теперь, select предлагает нам выбрать одно из значений, это значение назначается переменной number, затем case, в зависимости от значения переменной, запускает соответствующую команду.

```
[user@centos8 ~]$ nano case
[user@centos8 ~]$ ./case
1) 1
2) 2
3) 3
4) stop
#? 1
One
#? 4
[user@centos8 ~]$
```

А чтобы не застрять в бесконечной петле, в списке опций пропишем что-нибудь типа stop, а в case используем команду break, чтобы прекратить цикл. После break цикл прерывается и начинают выполняться другие команды после цикла - ./case.

```
else
  echo Welcome!
  select option in "Add user" "Show users" "Exit"
  do case $option in
    "Add user") read -p "Print username: " user
                read -p "Print groupname: " group
                create_user;;
    "Show users") cut -d: -f1 /etc/passwd ;;
    "Exit") break ;;
    *) echo Wrong option ;;
  esac
done
fi
```

Теперь добавим это в наш скрипт. Допустим, сделаем так, чтобы можно было добавить пользователя, посмотреть текущих пользователей, либо выйти - select option in "Add user" "Show users" "Exit" do case \$option in "Add user") read -p ... ;; "Show users") cut -d: -f1 /etc/passwd ;; "Exit") break ;; *) echo Wrong option ;; esac ;; done.

```
[user@centos8 ~]$ sudo rm /var/users
[user@centos8 ~]$ sudo ./myscript
Welcome!
1) Add user
2) Show users
3) Exit
#? 1
Print username: user36
Print groupname: it
groupadd: group 'it' already exists
%it ALL=(ALL) ALL
mkdir: cannot create directory '/home/it': File exists
#? 2
root
bin
daemon
adm
lp
```

Сохраним, удалим файл - `sudo rm /var/users` - чтобы наш скрипт предложил интерактивное меню и попробуем запустить скрипт - `sudo ./myscript`. Выбираем опцию 1 – у нас выходит приглашение ввести имя пользователя и группу. Окей, нажимаем `enter` – меню появилось ещё раз. Теперь выбираем 2 – и видим список всех пользователей. Выбираем 3 и выходим.

Подводя итоги, сегодня мы с вами разобрали команду `for`, с помощью которой мы можем создавать циклы; переменную `IFS`; функции, с помощью которых можем вызывать одну или несколько заранее прописанных команд; команду `select`, с помощью которой мы можем создать интерактивное меню; и команду `case`, с помощью которой мы можем запускать команды в зависимости от значения переменной.