Продолжение условного оператора if. Вложенные условия

Условный оператор — это конструкция, которая позволяет проверить условие, и если оно выполняется (в нашем случае возвращает 0), выполнить какое-то действие. В противном случае — выполнить другое действие.

Для этого используются ключевые слова if, then, else. Чтобы обозначить окончание условного оператора, используется ключевое слово fi. Отступы здесь не несут никакого функционального значения, но они помогают сделать наш скрипт более читаемым.
Задавать else, то есть действие, выполняемое в случае, когда условие не выполняется, не обязательно.

Пример использования elif

Давайте вернёмся к нашему скрипту, который проверяет, не Новый ли год сегодня. И посмотрим на его примере использование elif. Предположим, мы хотели бы добавить ещё какую-нибудь проверку, например не Восьмое ли марта сегодня.

Мы могли бы добавить ещё один такой же оператор, который сравнивает сегодняшнюю дату с 08.03, но теперь мы знаем способ сделать это же, но в более лаконичной записи.

```
if [ $today =31.12 ]
then
  echo "Happy New Year!"
elif [ $today = 08.03 ]
then
  echo "Happy International Women's Day!"
else
  echo "Today is $today"
fi
```

Если ничего из этого не выполняется, то напоминаем, какое сегодня число. Обратите внимание, что мы дописали условие в уже существующий оператор, вместо того чтобы писать новый. Так что, например, мы можем не переписывать ту часть оператора, которая выводит сообщение с напоминанием о сегодняшнем числе.

Конструкция elif позволяет нам последовательно проверять несколько условий.

Скрипт для проверки логина и пароля в bash

Давайте напишем небольшой скрипт, который будет предлагать нам представиться, спросит пароль и проверит его.

Откроем редактор Vim и напишем в первой строчке шебанг и путь до bash.

```
#!/bin/bash
```

Введём пользователя, его логин.

```
echo "Please enter a username"
```

И дальше будем проверять, что имя пользователя именно то, которое мы ожидаем, при помощи условного оператора. Но как нам «забрать» то, что ввёл пользователь? Сохраним скрипт и пока закроем.

Команда шелла *read* читает стандартный ввод, превращает его в строку и сохраняет в переменную, которая передаётся ей как параметр.

То есть мы могли бы сделать вот так, передаём команде *read* имя переменной:

```
read season
```

Вводим то, что хотим поместить в эту переменную:

```
fall
```

И теперь выведем значение этой переменной:

```
echo $season
```

Чтобы узнать побольше о встроенных командах, используется команда *help*. Определять, встроенная команда перед нами или отдельная программа, мы уже умеем:

```
type -a read
help read
```

Отсюда можно узнать несколько интересных вещей. Например, если не передать название переменной, наш ввод сохранится в переменную, которая задана по умолчанию. Она называется REPLY:

```
read
zzz
echo $REPLY
```

Ещё команда *read* умеет и сама выводить промпт, то есть приглашение на ввод какой-то строки, если указать параметр -p:

```
read -p "Please enter something: " line
echo $line
```

Если бы мы не указали имя переменной, введённую строку мы бы могли получить, посмотрев содержимое переменной $REPLY.

Ещё из справочной информации, которую выводит *help,* мы можем узнать, что *read* умеет разбивать введённую строку по пробелам и записывать ввод в несколько переменных (если передано несколько имён).

Посмотрим на более простом примере:

```
read aa bb cc
1 2 3
echo cc
```

Если вы введёте больше слов, чем названий переменных, все оставшиеся слова запишутся в последнюю переменную.

```
read aa bb
1 2 3
echo $bb
```

Отдельно выводить приглашение и отдельно считывать то, что введёт пользователь, нам не нужно, поэтому напишем:

```
read -p "Please enter username: " username
```

Выведем промпт и считаем ввод в переменную.

Так как мы ожидаем определённое имя, нам нужно сравнить введённую строку с именем, которое известно нашему скрипту:

```
if [ "$username" = "Anna" ]; then
```

и если пользователь ввёл правильное имя, его нужно попросить ввести пароль:

```
read -p "Please enter password: " password
```

Теперь нам нужно проверить введённый пароль. В тех случаях, которые мы уже встречали, после then идёт else, то есть действие в случае, когда условие не выполняется. То есть мы можем написать:

```
else echo "Invalid Username!"
fi
```

И в следующем условном операторе проверять, верен ли введённый пароль.

То есть, если условие выполняется, после ключевого слова then мы можем написать не только какие-то действия (вывести строку, скопировать файл), но и новый оператор if.

Теперь после запроса на ввод пароля напишем вложенный условный оператор. Синтаксис ничем не отличается от обычного:

```
if [ $password = "123" ]
  then echo "Hello, Anna!"
else
  echo "Invalid Password!"
  exit 1
fi
```

Здесь помогают отступы. Можно увидеть, где начинается и где заканчивается вложенный условный оператор.