

The test command

```
$ test 10 -lt 5
$ echo $?
1                # "False", "Failure"
$ test 10 -gt 5
$ echo $?
0                # "True", "Success"
```

- Another syntax for the test command:
Don't forget the space after [and before]

```
$ [ 10 -lt 5 ]
$ echo $?
1                # "False", "Failure"
$ [ 10 -gt 5 ]
$ echo $?
0                # "True", "Success"
```

test operators

comparison operator	description
=, !=, <, >	compares two <u>string</u> variables
-z, -n	tests if a string is empty (zero-length) or not empty (nonzero-length)
-lt, -le, -eq, -gt, -ge, -ne	compares <u>numbers</u> ; equivalent to Java's <, <=, ==, >, >=, !=
-e, -f, -d	tests whether a given file or directory exists
-r, -w, -x	tests whether a file exists and is readable/writable/executable

```
if [ $USER = "husky14" ]; then
```

```
    echo 'Woof! Go Huskies!'
```

```
fi
```

```
LOGINS=$(w -h | wc -l)
```

```
if [ $LOGINS -gt 10 ]; then
```

```
    echo 'server is very busy right now!'
```

```
fi
```

BMI Exercise

- Write a program that computes the user's body mass index (BMI) to the nearest integer, as well as the user's weight class:

$$BMI = \frac{weight}{height^2} \times 703$$

BMI	Weight class
≤ 18	underweight
18 - 24	normal
25 - 29	overweight
≥ 30	obese

```
$ ./bmi.sh
```

```
Usage: ./bmi.sh weight height
```

```
$ ./bmi.sh 112 72
```

```
Your Body Mass Index (BMI) is 15
```

```
Here is a sandwich; please eat.
```

```
$ ./bmi.sh 208 67
```

```
Your Body Mass Index (BMI) is 32
```

```
There is more of you to love.
```

BMI Exercise solution

```
#!/bin/bash
# Body Mass Index (BMI) calculator
if [ $# -lt 2 ]; then
    echo "Usage: $0 weight height"
    exit 1          # 1 indicates failure, 0 for success
fi

let H2="$2 * $2"
let BMI="703 * $1 / $H2"
echo "Your Body Mass Index (BMI) is $BMI"
if [ $BMI -le 18 ]; then
    echo "Here is a sandwich; please eat."
elif [ $BMI -le 24 ]; then
    echo "You're in normal weight range."
elif [ $BMI -le 29 ]; then
    echo "You could lose a few."
else
    echo "There is more of you to love."
fi
```

Common errors

- `[: -eq :` unary operator expected
 - you used an undefined variable in an `if` test
- `[:` too many arguments
 - you tried to use a variable with a large, complex value (such as multi-line output from a program) as though it were a simple `int` or `string`
- `let :` syntax error: operand expected (error token is " ")
 - you used an undefined variable in a `let` mathematical expression

while and until loops

```
while [ condition ]; do    # go while condition is true
    commands
done
```

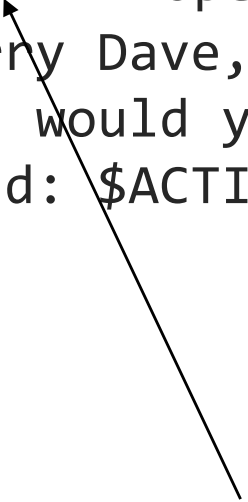
```
until [ condition ]; do    # go while condition is false
    commands
done
```

While exercise

- Prompt the user for what they would like to do. While their answer is “open the pod bay doors” tell them that you cannot do that and prompt for another action.

While Exercise solution

```
#!/bin/bash
# What would you like to do?
read -p "What would you like me to do? " ACTION
echo "You said: $ACTION"
while [ "$ACTION" == "open the pod bay doors" ]; do
    echo "I'm sorry Dave, I'm afraid I can't do that."
    read -p "What would you like me to do? " ACTION
    echo "You said: $ACTION"
done
echo "Bye"
```



The quotes around "\$ACTION" are important here,
try removing them and see what happens.

select and case

- Bash Select statement:

```
PS3=prompt # Special variable* for the select prompt
select choice in choices; do
    commands
    break # Break, otherwise endless loop
done
```

- Bash Case statement:

```
case EXPRESSION in
    CASE1) COMMAND-LIST;;
    CASE2) COMMAND-LIST;;
    ...
    CASEN) COMMAND-LIST;;
esac
```

Select Example

```
PS3="What is your favorite food? " # Goes with the select stmt
```

```
echo "Welcome to the select example!"
```

```
echo "It prints out a list of choices"
```

```
echo "but does nothing interesting with the answer."
```

```
select CHOICE in "pizza" "sushi" "oatmeal" "broccoli"; do
```

```
    echo "You picked $CHOICE"
```

```
    break
```

```
done
```

```
echo "For the select statement, you pick a number as your choice."
```

Case Example

```
echo "Welcome to the case example!"  
echo "Without a select statement, you must get the spelling/case exact."  
read -p "What format do you prefer? (tape/cd/mp3/lp) " FORMAT  
echo "You said $FORMAT"
```

```
case "$FORMAT" in  
    "tape") echo "no random access!";;  
    "cd") echo "old school";;  
    "mp3") echo "how modern";;  
    "lp") echo "total retro";;  
esac
```

select/case Exercise

- Have the user select their favorite kind of music, and output a message based on their choice

select/case Exercise Solution

```
PS3="What is your favorite kind of music? "  
select CHOICE in "rock" "pop" "dance" "reggae"; do  
    case "$CHOICE" in  
        "rock") echo "Rock on, dude.>";;  
        "pop") echo "Top 100 is called that for a reason.>";;  
        "dance") echo "Let's lay down the Persian!>";;  
        "reggae") echo "Takin' it easy...>";;  
        * ) echo "come on...you gotta like something!>";;  
    esac  
    break  
done
```

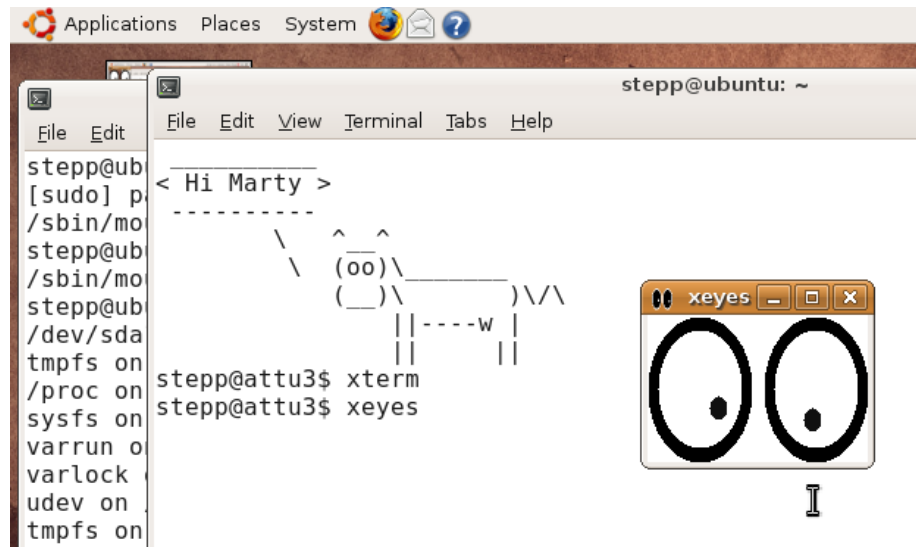
Other useful tidbits

Remote X display

Normally, you **can't** run graphical programs on **remote** servers however, if you connect your SSH with the **-X** parameter, you can!

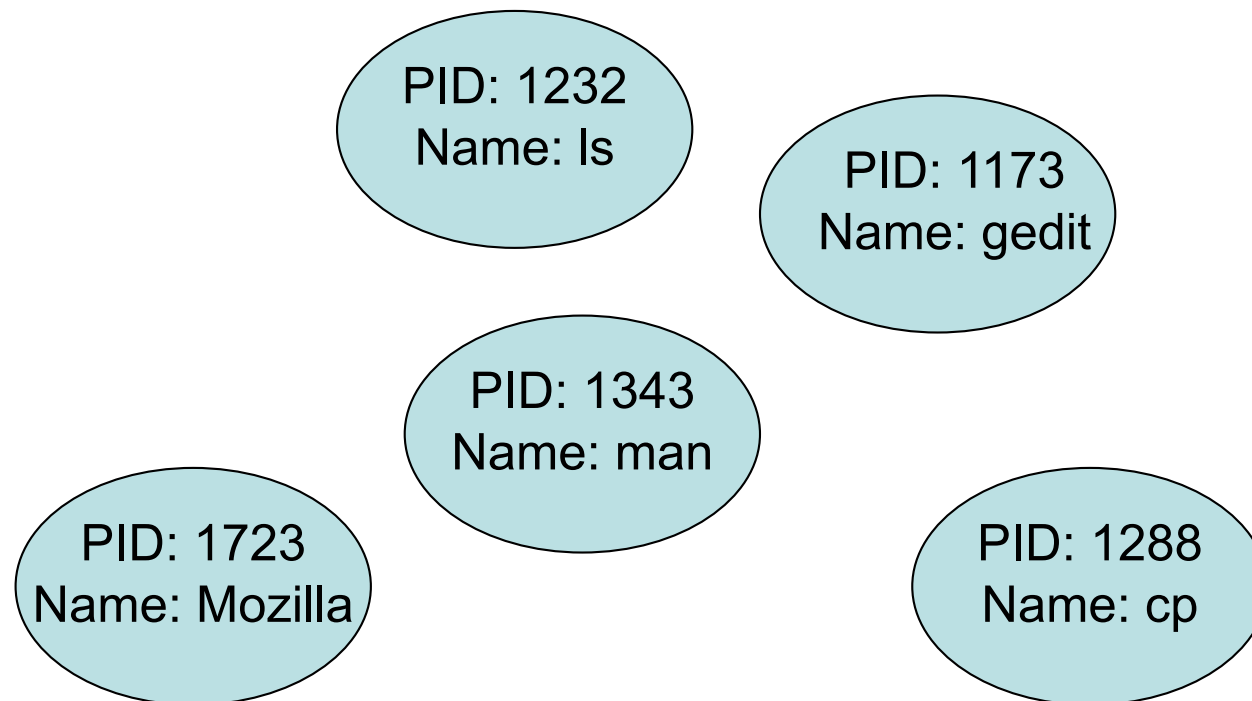
- the X-Windows protocol is capable of displaying programs remotely

`ssh -X username@ip_address`



Processes

- **process:** a program that is running (essentially)
 - when you run commands in a shell, it launches a process for each command
 - Process management is one of the major purposes of an OS



Process commands

command	description
ps or jobs	list processes being run by a user; each process has a unique integer id (PID)
top	show which processes are using CPU/memory; also shows stats about the computer
kill	terminate a process by PID (sometimes -KILL is needed)
killall	terminate several processes by name

- use `kill` or `killall` to stop a runaway process (infinite loop)

Connecting with ssh

command	description
ssh	open a shell on a remote server

- Linux/Unix are built to be used in multi-user environments where several users are logged in to the same machine at the same time
 - users can be logged in either locally or via the network
- You can connect to other Linux/Unix servers with ssh
 - once connected, you can run commands on the remote server
 - other users might also be connected; you can interact with them
 - can connect even from other operating systems

```
$ ssh username@address
```

Network commands

command	description
<code>links</code> or <code>lynx</code>	text-only web browsers (really!)
<code>ssh</code>	connect to a remote server
<code>sftp</code> or <code>scp</code>	transfer files to/from a remote server (after starting sftp, use <code>get</code> and <code>put</code> commands)
<code>wget</code>	download from a URL to a file
<code>curl</code>	download from a URL and output to console
<code>alpine</code> , <code>mail</code>	text-only email programs