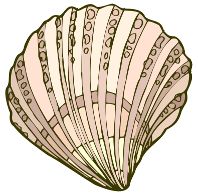
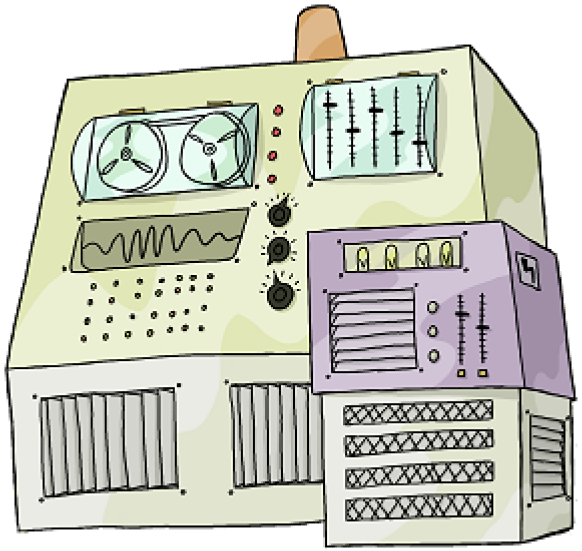
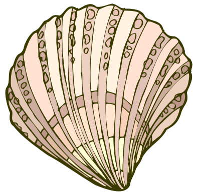
The Unix Shell

Job Control



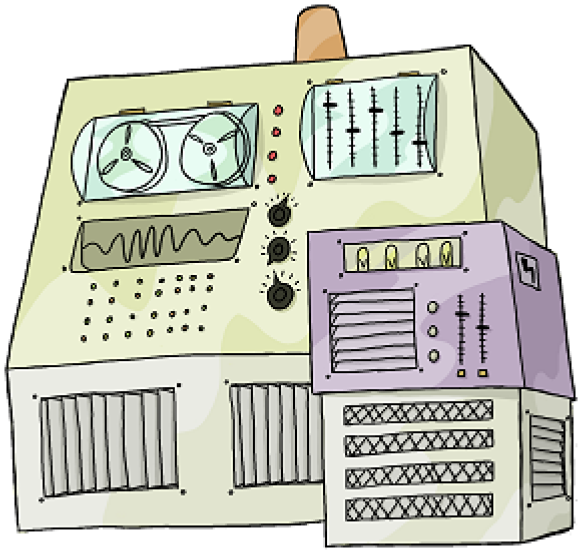
shell

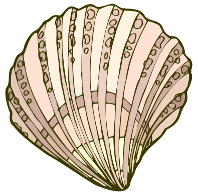




shell

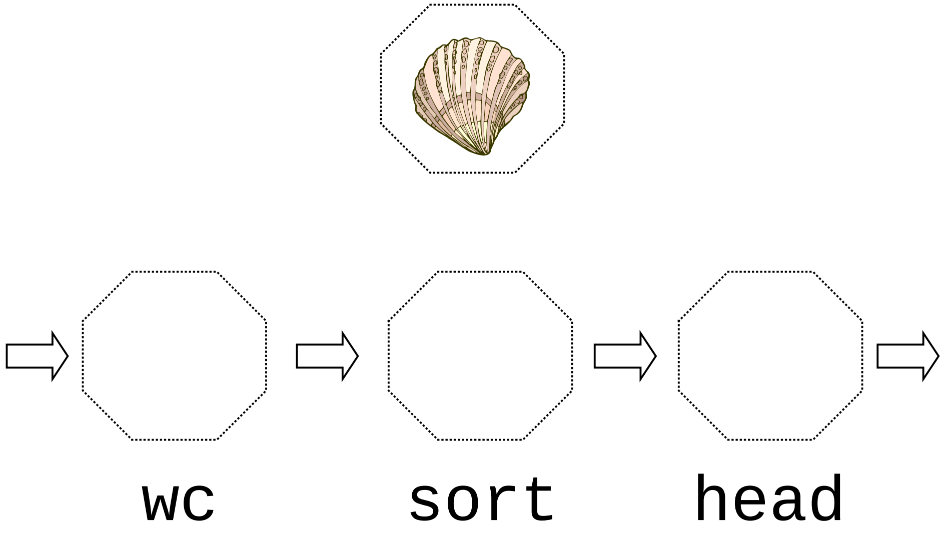
```
$ wc –l *.pdb | sort -n | head -n 1
```

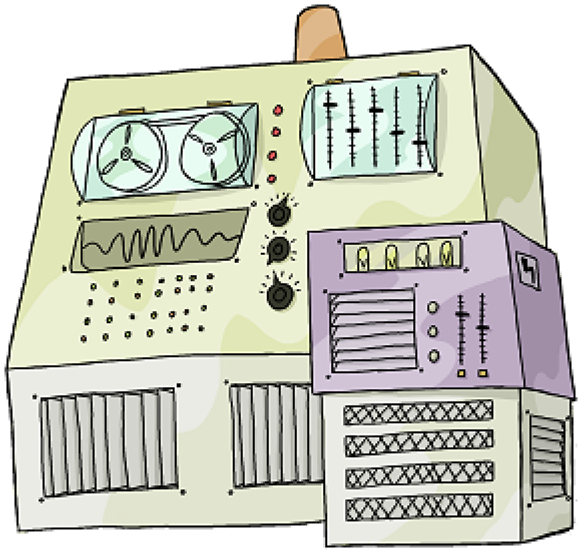


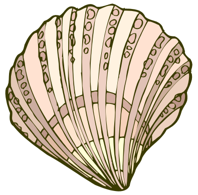


shell

```
$ wc –l *.pdb | sort -n | head -n 1
```

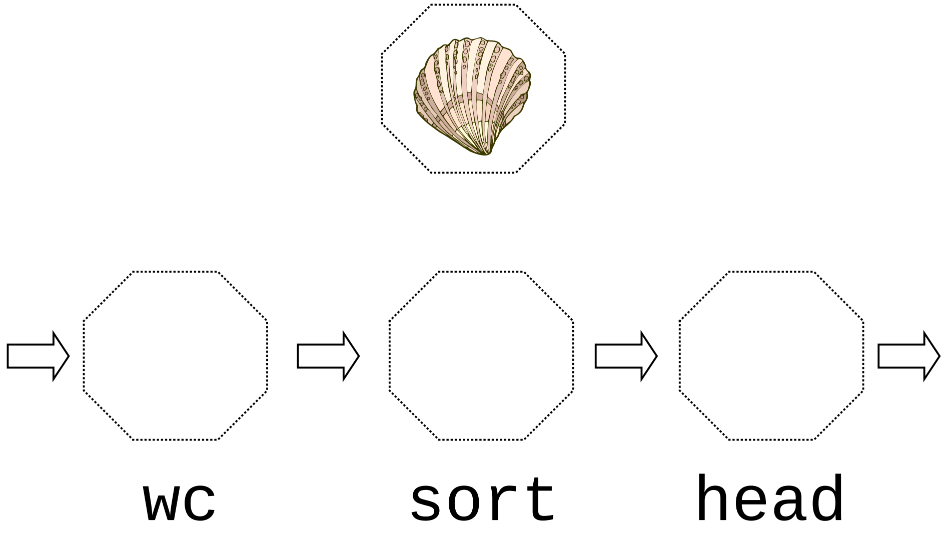


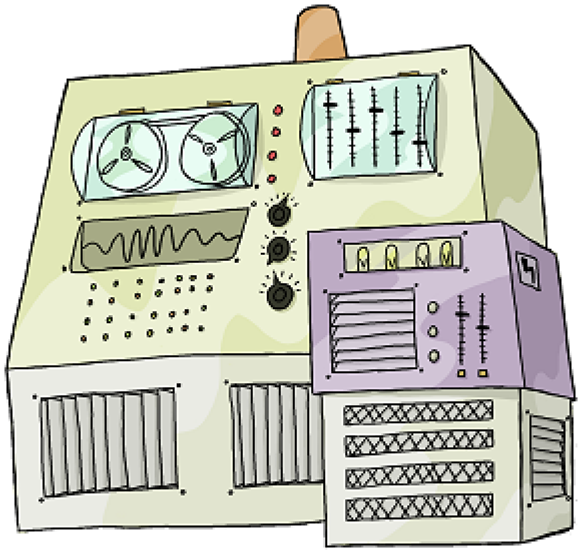




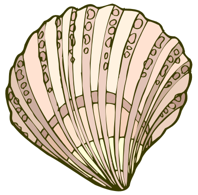
shell

```
$ wc –l *.pdb | sort -n | head -n 1
```



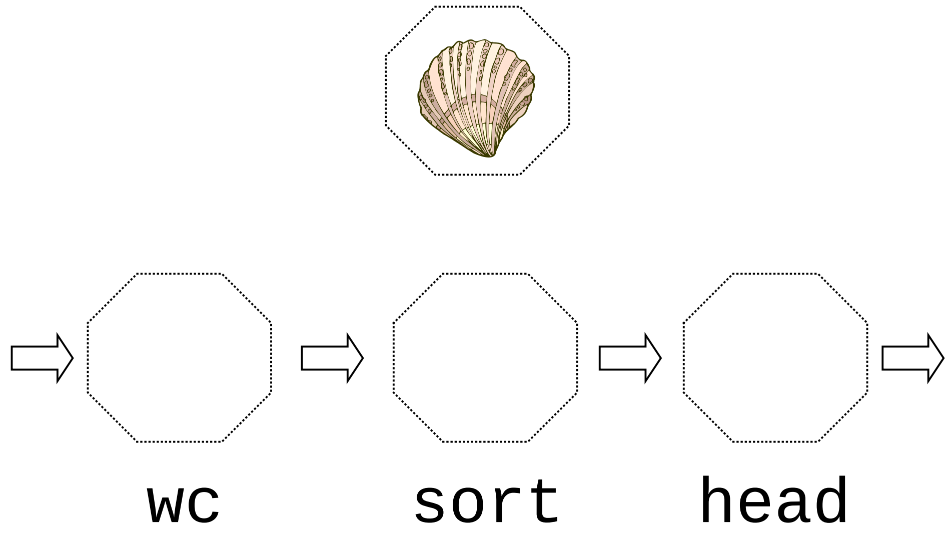


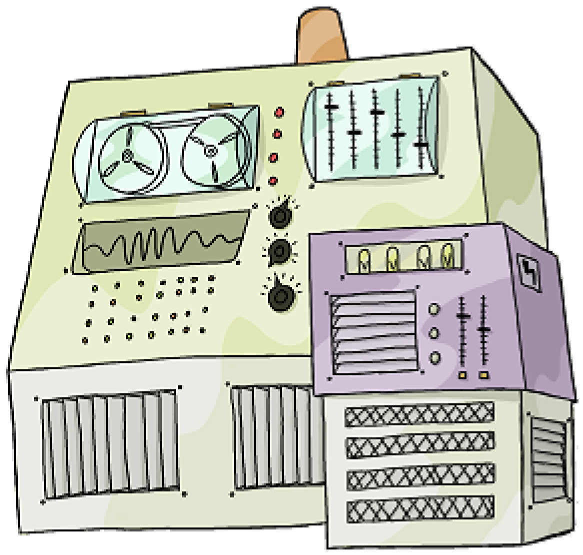
Control programs while they run



shell

```
$ wc –l *.pdb | sort -n | head -n 1
```





processes

Control ~~programs~~ while they run

A *process* is a running program

A *process* is a running program

Some are yours

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use `ps` to get a list

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use ps to get a list

```
$ ps
  PID TTY          TIME CMD
11275 pts/16   00:00:00 bash
12092 pts/16   00:00:00 ps
```

Command

Process ID

A *process* is a running program

Some are yours

Most belong to the operating system (or other users)

Use ps to get a list (in various formats)

See "man ps"

```
$ ps
  PID TTY          TIME CMD
11275 pts/16   00:00:00 bash
12092 pts/16   00:00:00 ps

$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
vlad     11275  0.0  0.0 108608  1856 pts/16   Ss   14:59   0:00 -bash
vlad     12096  0.0  0.0 108320  1016 pts/16   R+   15:03   0:00 ps ux

$ ps -F
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
vlad     11275 11224  0 27152  1856   1 14:59 pts/16   00:00:00 -bash
vlad     12104 11275  0 27079  1016   5 15:03 pts/16   00:00:00 ps -F
```

Can stop, pause, and resume running processes

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$
```

Stop the running program

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$
```

Run in the background

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat $\boxed{\&}$
$
```

Run in the background

Shell returns right away instead

of waiting for the program to finish

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$
```

Can run other programs in the *foreground*

while waiting for background process(es) to finish

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$
```

Show background processes

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$ fg
```

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$ fg
```

Bring background job to foreground

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$ fg
```

Bring background job to foreground

Use `fg %1`, `fg %2`, etc. if there are several background jobs

Can stop, pause, and resume running processes

```
$ ./analyze results*.dat
...a few minutes pass...
^C
$ ./analyze results*.dat &
$ fbcmd events
$ jobs
[1] ./analyze results01.dat results02.dat results03.dat
$ fg
...a few minutes pass...
$
```

$\xleftarrow{\hspace{4em}}$ And finally it's done

Use ^Z to pause a program that's already running

Use ^Z to pause a program that's already running

`fg` to resume it in the foreground

Use ^Z to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

Use ^Z to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
```

Use ^Z to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
^Z
[1]  Stopped   ./analyze results01.dat
$
```

Use ^Z to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
^Z
[1]  Stopped   ./analyze results01.dat
$ bg %1
$
```

Use ^Z to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
^Z
[1]  Stopped   ./analyze results01.dat
$ bg %1
$ jobs
[1] ./analyze results01.dat
$
```

Use ^Z to pause a program that's already running

`fg` to resume it in the foreground

Or `bg` to resume it as a background job

```
$ ./analyze results01.dat
^Z
[1]  Stopped   ./analyze results01.dat
$ bg %1
$ jobs
[1] ./analyze results01.dat
$ kill %1
$
```

Job control mattered a lot when users only had one terminal window

Job control mattered a lot when users only had one terminal window

Less important now: just open another window

Job control mattered a lot when users only had one terminal window

Less important now: just open another window

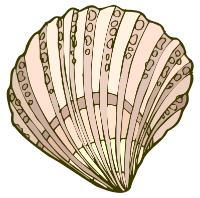
Still useful when running programs remotely created by

Greg Wilson

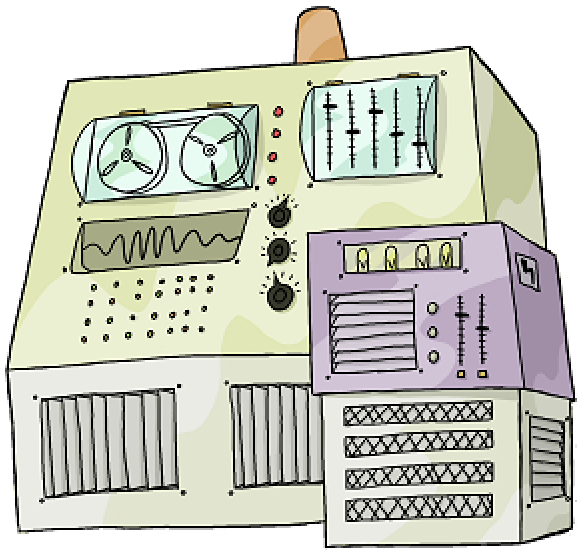
August 2010

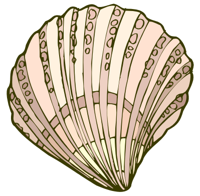
Job Control

Variables



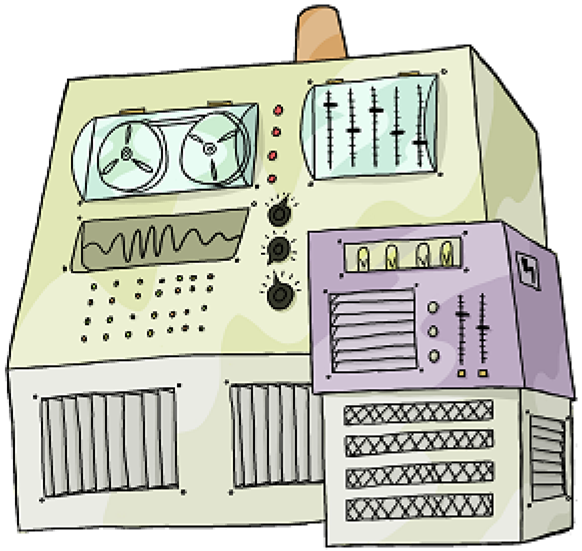
shell

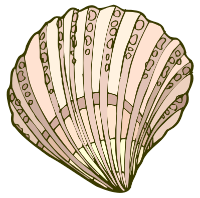




shell

The shell is a program

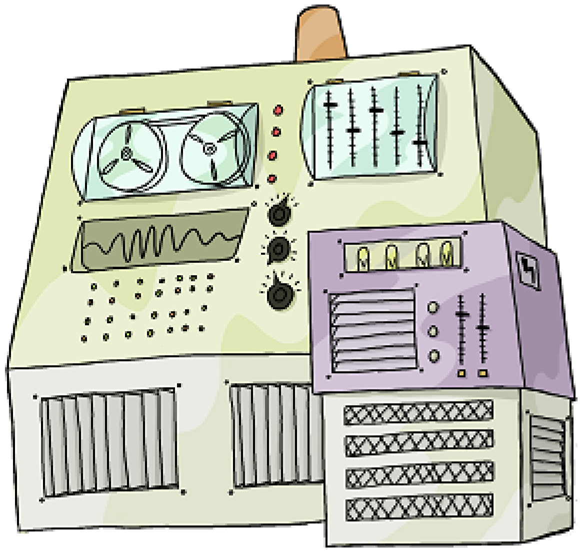


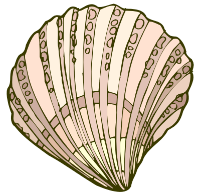


shell

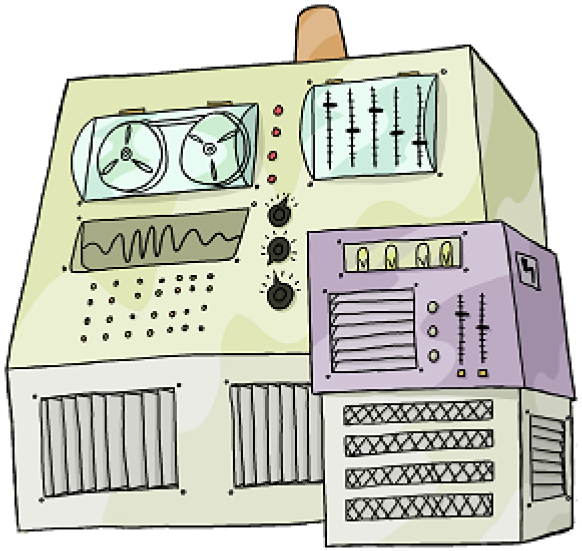
The shell is a program

It has variables





shell



The shell is a program

It has variables

Changing their values

changes its behavior

```
$ set
COMPUTERNAME=TURING
HOME=/home/vlad
HOMEDRIVE=C:
HOSTNAME=TURING
HOSTTYPE=i686
MANPATH=/usr/local/man:/usr/share/man:/usr/man
NUMBER_OF_PROCESSORS=4
OS=Windows_NT
PATH=/usr/local/bin:/usr/bin:/bin
PWD=/home/vlad
UID=1000
USERNAME=vlad
```

```
$ set
COMPUTERNAME=TURING
HOME=/home/vlad
HOMEDRIVE=C:
HOSTNAME=TURING
HOSTTYPE=i686
MANPATH=/usr/local/man:/usr/share/man:/usr/man
NUMBER_OF_PROCESSORS=4
OS=Windows_NT
PATH=/usr/local/bin:/usr/bin:/bin
PWD=/home/vlad
UID=1000
USERNAME=vlad
```

With no arguments, shows all variables and their values

```
$ set
COMPUTERNAME=TURING
HOME=/home/vlad
HOMEDRIVE=C:
HOSTNAME=TURING
HOSTTYPE=i686
MANPATH=/usr/local/man:/usr/share/man:/usr/man
NUMBER_OF_PROCESSORS=4
OS=Windows_NT
PATH=/usr/local/bin:/usr/bin:/bin
PWD=/home/vlad
UID=1000
USERNAME=vlad
```

Standard to use upper-case names

```
$ set
COMPUTERNAME=TURING
HOME=/home/vlad
HOMEDRIVE=C:
HOSTNAME=TURING
HOSTTYPE=i686
MANPATH=/usr/local/man:/usr/share/man:/usr/man
NUMBER_OF_PROCESSORS=4
OS=Windows_NT
PATH=/usr/local/bin:/usr/bin:/bin
PWD=/home/vlad
UID=1000
USERNAME=vlad
```

All values are strings

```
$ set
COMPUTERNAME=TURING
HOME=/home/vlad
HOMEDRIVE=C:
HOSTNAME=TURING
HOSTTYPE=i686
MANPATH=/usr/local/man:/usr/share/man:/usr/man
NUMBER_OF_PROCESSORS=4
OS=Windows_NT
PATH=/usr/local/bin:/usr/bin:/bin
PWD=/home/vlad
UID=1000
USERNAME=vlad
```

All values are strings

Programs must convert to other types when/as necessary

```
$ set
COMPUTERNAME=TURING
HOME=/home/vlad
HOMEDRIVE=C:
HOSTNAME=TURING
HOSTTYPE=i686
MANPATH=/usr/local/man:/usr/share/man:/usr/man
NUMBER_OF_PROCESSORS=4
OS=Windows_NT
PATH=/usr/local/bin:/usr/bin:/bin
PWD=/home/vlad
UID=1000
USERNAME=vlad
```

`int(string)` for numbers

```
$ set
COMPUTERNAME=TURING
HOME=/home/vlad
HOMEDRIVE=C:
HOSTNAME=TURING
HOSTTYPE=i686
MANPATH=/usr/local/man:/usr/share/man:/usr/man
NUMBER_OF_PROCESSORS=4
OS=Windows_NT
PATH=/usr/local/bin:/usr/bin:/bin
PWD=/home/vlad
UID=1000
USERNAME=vlad
```

`split(':')` for lists

PATH controls where the shell looks for programs

PATH controls where the shell looks for programs

```
$ ./analyze
```

Run the `analyze` program

in the current directory

PATH controls where the shell looks for programs

```
$ ./analyze
$ /bin/analyze
```

$\xleftarrow{}$ Run the `analyze` program in the /bin directory

PATH controls where the shell looks for programs

```
$ ./analyze
$ /bin/analyze
$ analyze
```

PATH controls where the shell looks for programs

```
$ ./analyze
$ /bin/analyze
$ analyze
```

```
directories = split(PATH, ':')
for each directory:
  if directory/analyze exists,
    run it
```

PATH controls where the shell looks for programs

```
$ ./analyze
$ /bin/analyze
$ analyze
```

```
directories = split(PATH, ':')
for each directory:
  if directory/analyze exists,
    run it
```

```
/usr/local/bin
/usr/bin
/bin
```

```
/usr/bin/analyze
```

PATH controls where the shell looks for programs

```
$ ./analyze
$ /bin/analyze
$ analyze
```

```
directories = split(PATH, ':')
for each directory:
  if directory/analyze exists,
    run it (and then stop searching)
```

```
/usr/local/bin
/usr/bin
/bin
```

```
/usr/bin/analyze

(/bin/analyze)
```

`echo` prints its arguments

`echo` prints its arguments

Use it to show variables' values

`echo` prints its arguments

Use it to show variables' values

```
$ echo hello transylvania
hello transylvania
$
```

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania
hello transylvania
$ echo HOME
```

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania
hello transylvania
$ echo HOME
HOME
$
```

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania
hello transylvania
$ echo HOME
HOME
$ echo $HOME
/home/vlad
$
```

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania
hello transylvania
$ echo HOME
HOME
$ echo $HOME
/home/vlad
$
```

Ask shell to replace variable name with value before program runs

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania
hello transylvania
$ echo HOME
HOME
$ echo $HOME
/home/vlad
$
```

Ask shell to replace variable name with value before program runs

Just like * and ? are expanded before the program runs

echo prints its arguments

Use it to show variables' values

```
$ echo hello transylvania
hello transylvania
$ echo HOME
HOME
$ echo $HOME  ——→  echo /home/vlad
/home/vlad
$
```

Create variable by assigning to it

Create variable by assigning to it

Change values by reassigning to existing variables

Create variable by assigning to it

Change values by reassigning to existing variables

```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ SECRET_IDENTITY=Camilla
$ echo $SECRET_IDENTITY
Camilla
$
```

Assignment only changes variable's value

in *this* shell

Assignment only changes variable's value in *this* shell

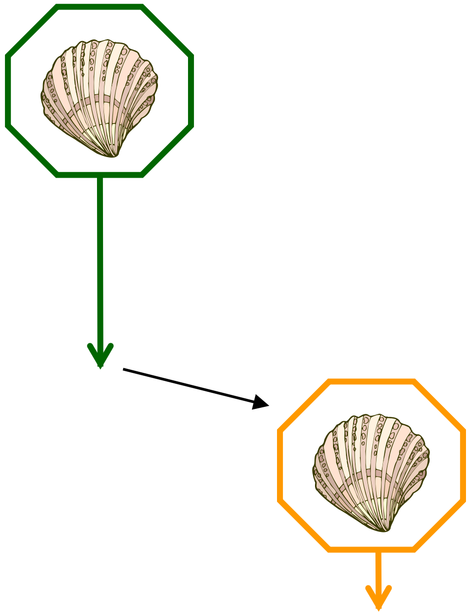
```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$
```

Assignment only changes variable's value in *this* shell

```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$
```

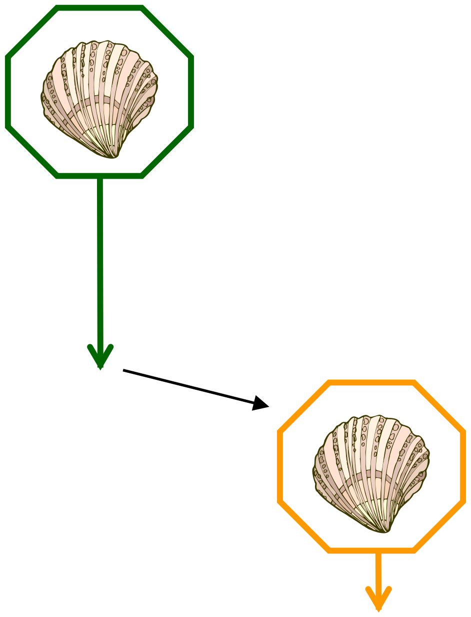
Assignment only changes variable's value in *this* shell

```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$
```



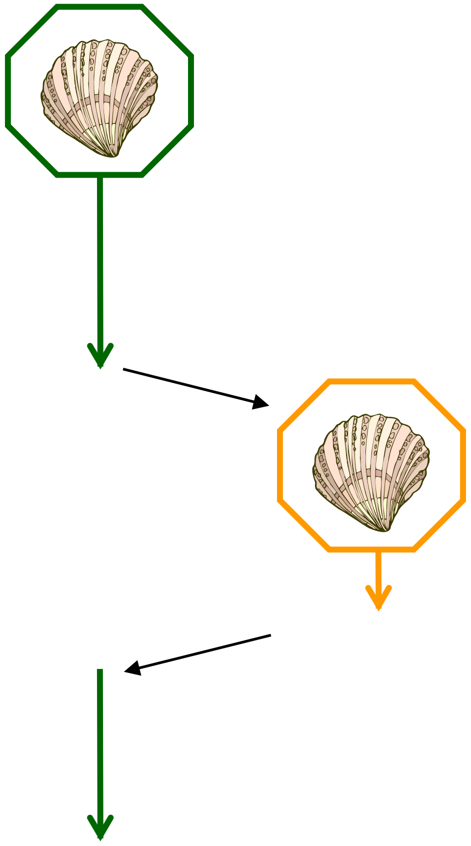
Assignment only changes variable's value in *this* shell

```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$ echo $SECRET_IDENTITY
$
```



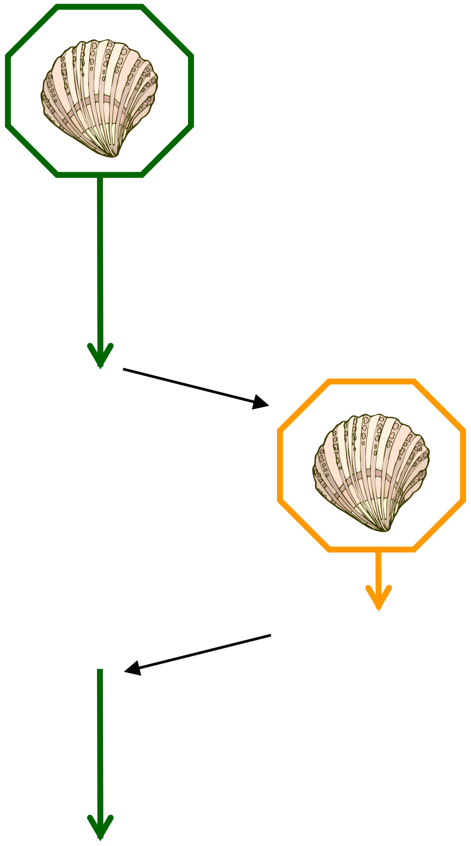
Assignment only changes variable's value in *this* shell

```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$ echo $SECRET_IDENTITY
$ exit
$
```



Assignment only changes variable's value in *this* shell

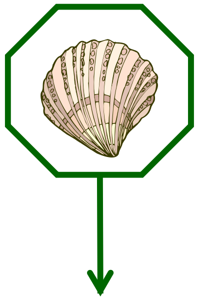
```
$ SECRET_IDENTITY=Dracula
$ echo $SECRET_IDENTITY
Dracula
$ bash
$ echo $SECRET_IDENTITY
$ exit
$ echo $SECRET_IDENTITY
Dracula
$
```



Use `export` to signal that the variable should be visible to subprocesses

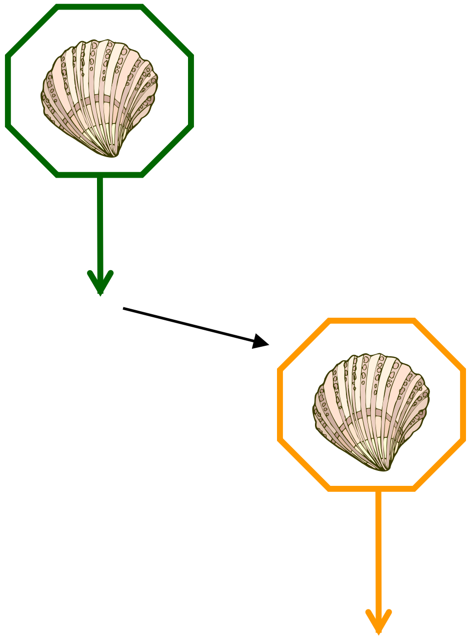
Use export to signal that the variable should be visible to subprocesses

```
$ SECRET_IDENTITY=Dracula
$ export SECRET_IDENTITY
$
```



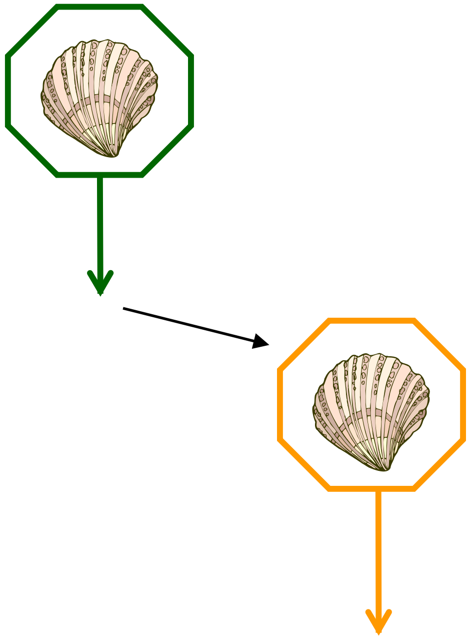
Use export to signal that the variable should be visible to subprocesses

```
$ SECRET_IDENTITY=Dracula
$ export SECRET_IDENTITY
$ bash
$
```



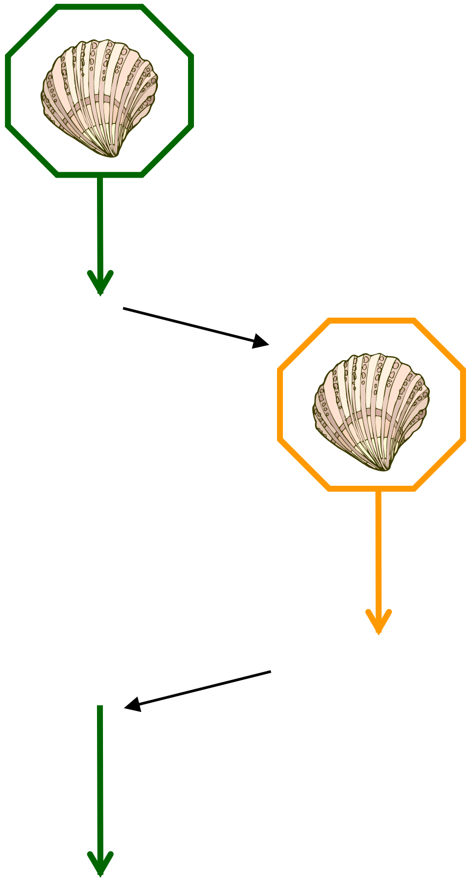
Use export to signal that the variable should be visible to subprocesses

```
$ SECRET_IDENTITY=Dracula
$ export SECRET_IDENTITY
$ bash
$ echo $SECRET_IDENTITY
Dracula
$
```



Use export to signal that the variable should be visible to subprocesses

```
$ SECRET_IDENTITY=Dracula
$ export SECRET_IDENTITY
$ bash
$ echo $SECRET_IDENTITY
Dracula
$ exit
$
```



Commands in `$HOME/.bashrc` are executed when shell starts

Commands in `$HOME/.bashrc` are executed when shell starts

```
export SECRET_IDENTITY=Dracula
export BACKUP_DIR=$HOME/backup
```

`/home/vlad/.bashrc`

Commands in `$HOME/.bashrc` are executed when shell starts

```
export SECRET_IDENTITY=Dracula
export BACKUP_DIR=$HOME/backup
```

Also common to use `alias` to create shortcuts

Commands in `$HOME/.bashrc` are executed when shell starts

```
export SECRET_IDENTITY=Dracula
export BACKUP_DIR=$HOME/backup
```

Also common to use `alias` to create shortcuts

```
alias backup=/bin/zarble -v --nostir -R 20000 $HOME $BACKUP_DIR
```

Commands in `$HOME/.bashrc` are executed

when shell starts

```
export SECRET_IDENTITY=Dracula
export BACKUP_DIR=$HOME/backup
```

Also common to use `alias` to create shortcuts

```
alias backup=/bin/zarble -v --nostir -R 20000 $HOME $BACKUP_DIR
```

Not something you want to type over and over created by

Greg Wilson

August 2010