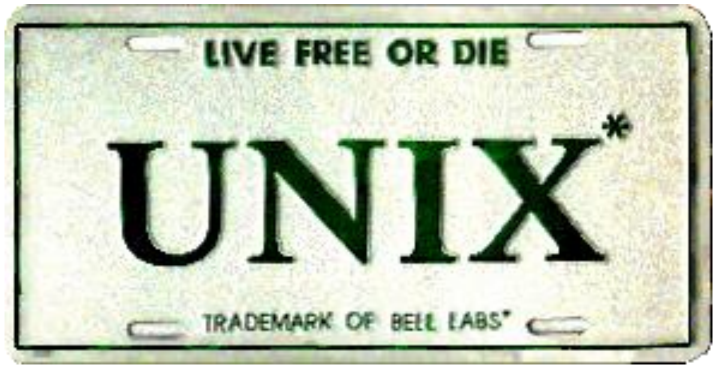


UNIX for Programmers and Users

- **Background Processing**

```
$ find  .   -name    a.c     --->search for "a.c"
./wild/a.c
./reverse/tmp/a.c

$ find  .   -name  b.c &    --->search in the background.
27174                        --->process ID number.

$ date                              -->run "date" in the foreground.
./wild/b.c                          -->output from background "find".
Wed, Aug 24, 2016  11:59:08 AM -->output from date.

$ ./reverse/tmp/b.c     -->more output from background "find"
                   -->came after we got the shell prompt,
                   --> but we don't get another prompt.
```

- **Background Processing**

- Several background commands may be specified on a single line by separating each command by an ampersand.

```
$ date & pwd &          ---> create two background processes.
27310                   ---> process ID of "date".
27311                   ---> process ID of "pwd".
/home/glass             ---> output from "pwd".
Wed, Aug 24, 2016  6:59:08 AM   ---> output from "date".
$ _
```

• REDIRECTING OUTPUT OF BACKGROUND PROCESSES

To prevent the output from a background process from arriving to the terminal, redirect its output to a file.

```
$ find  .   -name  a.c  > find.txt  &
27188                                   ---> process ID of “find”.

$ ls -l  find.txt                       ---> look at “find.txt”.
-rw-r--r--   1   glass     0  Aug   23   18:11   find.txt

$ ls -l  find.txt                       ---> watch it grow.
-rw-r--r--   1   glass    29  Aug   23   18:11   find.txt

$ cat  find.txt                         ---> list “find.txt”.
./wild/a.c
./reverse/tmp/a.c
$ _
```

- **SUBSHELLS**

1) When a grouped command such as (ls; pwd; date) is executed

   If the command is not executed in the background, the parent shell sleeps until the child shell terminates.

2) When a script is executed

   If the script is not executed in the background, the parent shell sleeps until the child shell terminates.

3) When a background job is executed

   The parent shell continues to run concurrently with the child shell.

- A child shell is called a *subshell*.

- *cd* commands executed in a subshell do not affect the working directory of the parent shell:

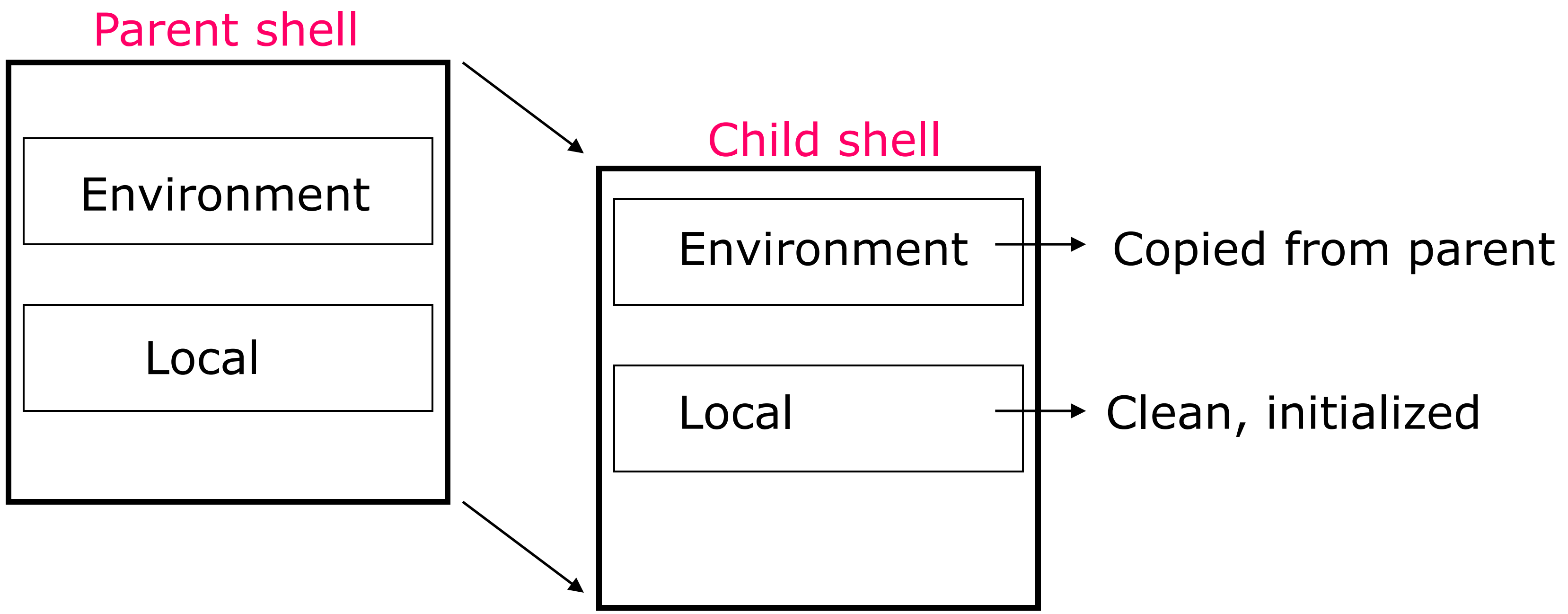
```
$ pwd              ---> display my login shell's current directory.
/home/glass

$ ( cd /; pwd ) ---> the subshell moves and executes pwd.
/                     ---> output comes from the subshell.

$ pwd               ---> my login shell never moved.
/home/glass
$ -
```

- Every shell contains two data areas:
  an environment space and a local-variable space.

  A child shell inherits a copy of its parent's environment space
  and a clean local-variable space:



Environment variables are therefore used for transmitting useful information between parent shells and their children.

- **VARIABLES**

- Here is a list of the predefined environment variables that are common to all shells:

Name	Meaning
$HOME	the full pathname of your home directory
$PATH	a list of directories to search for commands
$USER	your username
$SHELL	the full pathname of your login shell
$TERM	the type of your terminal

- **VARIABLES**

- the syntax for assigning a variable is as follows:

```
    variableName=value        ---> place no spaces around the value
    or
    variableName=" value "  ---> here, spacing doesn't matter.
```

```
$ echo $HOME
/home/SRD

$ HOME=UG1

$ echo $HOME
UG1
$
```

- **VARIABLES**

- The next example illustrates the difference between local and environment variables.

  In the following, we assign values to two local variables and then make one of them an environment variable by using the Bourne shell *export* command.

  Note that the value of the environment variable is copied into the child shell, but the value of the local variable is not.

  Finally, we press ***Control-D*** to terminate the child shell and restart the parent shell, and then display the original variables:

```
$ firstname="Shiv Ram"                 ---> set a local variable.
$ lastname=Dubey                       ---> set another local variable.
$ echo $firstname  $lastname           ---> display their values.
Shiv Ram Dubey

$ export lastname                      ---> make “lastname” an
                                       ---> environment variable.
$ bash                    ---> start a child shell; the parent sleeps.

$ echo  $firstname   $lastname         ---> display values again.
Dubey                     ---> note that firstname was’t copied.

$ ^D
$ echo  $firstname  $lastname          ---> they remain unchanged.
Shiv Ram Dubey
$ _
```