Real–time Systems
Shell Programming

[2] Shell Programming

Shell, command interpreter, is a program started up after opening of the user session by the **login** process. The shell is active till the occurence of the `<EOT>` character, which signals requests for termination of the execution and for informing about that fact the operating system kernel.

Each user obtains their own separate instance of the *sh*. Program *sh*prints out the monit on the screen showing its readiness to read a next command.

The shell interpreter works based on the following scenario:

1. displays a prompt,
2. waits for entering text from a keyboard,
3. analyses the command line and finds a command,
4. submit to the kernel execution of the command,
5. accepts an answer from the kernel and again waits for user input.

[3] The Shell Initialization

The shell initialization steps:

1. assigned values to **environmental variables**,
2. system scripts, defining other shell variables, executed,

	shell	system scripts
1.	sh, ksh	.profile
2.	csh	.login, .cshrc

Extended list of system scripts (stratup files) for the **bash** interpreter:

1. /etc/profile
2. /.bash_profile
3. /.bash_login
4. /.profile
5. **/.bashrc**
6. /.bash_logout

[4] **Users in the Unix System**

Users in the Unix system

- superuser, root,
- others users.

Users in the system, the /etc/passwd file:

- user name,
- password,
- uid, user identification,
- gid, group identification,
- GECOS field, only for informational purpose,
- the user's $HOME directory,
- the program to run at login.

```
tnowak:encrypted password:201:50::/usr/tnowak:/bin/sh
```

[5] **The /etc/shadow File**

The /etc/shadow file contains the encrypted password information for user's accounts and optional the password aging information.

- login name
- encrypted password
- days since Jan 1, 1970 that password was last changed
- days before password may be changed
- days after which password must be changed
- days before password is to expire that user is warned
- days after password expires that account is disabled
- days since Jan 1, 1970 that account is disabled
- a reserved field

[6] **User Groups**

A group in the system, the /etc/group file:

- group name,
- group password,
- group id,
- list of users belonging to the group.

```
wheel::10:tnowak,tkruk
```

Other issues:

- access rights to files (-rwxr-xr-x, 0755, the **chmod** command),
- SUID, SGID bits (-r-s–x–x, ccnp. the **passwd** command),

[7] The Shell Environment

There may be distinguished the following variables:

- **predefined variables**,
- **positional and special parameters**, relating to name and arguments of currently submitted command.

Exemplary variables of the **sh** shell:

HOME standard directory for the **cd** command,

IFS (ang. *Internal Field Separators*) used for word splitting after expansion and to split lines into words with the **read** builtin command

MAIL mailbox file with alerting on the arrival of the new mail,

PATH a colon-separated search path for commands

PS1 (prompt string 1), the primary prompt sting, under the **sh** shell: **$**,

PS2 (prompt string 2), the secondary prompt string, under the **sh** shell: >,

SHELL default program to be run as a subshell,

TERM a terminal type name. Identifies a set of steering sequences appropriate for some particular terminal (exemplary names: *ansi*, *vt100*, *xterm*),

[8] Commands

Submitting a command

```
$ [ VAR=value ... ] command_name [ arguments ... ]
$ echo $PATH
```

Built-in commands

```
$ PATH=$PATH:/usr/local/bin
$ export PATH
```

- the **set** built-in without any parameters prints values of all variables,
- the **export** built-in without any parameters prints values of all exported environmental variables.

[9] **Special Parameters**

Special parameters, these parameters may only be referenced, direct assignment to them is not allowed.

$0 name of the command

$1 first argument of the scipt/ function

$2 second argument of the script/ function

$9 ninth argument of the scipt/ function

$* all positional arguments `"$*" = "$1 $2 .."`

$@ list of separated all positional arguments `"$@" = "$1" "$2" ..`

$# the number of arguments of some commands or given to the last **set**,

$? exit status of the most recently executed foreground command,

$! PID of the most recently started backgruond command.

$$ PID of the current shell,

$0-9 also: may be set by the **set** command.

[10] **Metacharacters**

During resolving of file names and grouping commands into bigger sets, special characters called metacharacters are used.

```
*          string without the "/" character,
?          any single character,
[ ]        one character from the given set,
[...-...] like [ ], with given scope from the first to the last,
[!..-...] any character except those within the given scope,
#          start of a comment,
\          escape character, preserves the literal value of the following
           character,
$          a value of a variable named with the following string,
;          commands separator,
` `        string in accent characters executed as a command with the stdout
           of the execution as a result of that quotation,
' '        preserves the literal value of each character within the quotes
" "        preserves the literal value of all characters within the quotes,
           with the exception of $,  `,  and \
```

[11] **Command interpretation**

Steps in command interpretation under the **sh** shell:

1. entering line of characters,
2. division of the line into sequence of words, based on the IFS value,
3. substitution 1: subsitution of ${name} strings with variables' values,

```
$ b=/usr/user
$ ls -l prog.* > ${b}3
```

4. substitution 2: substitution of metacharacters * ? [ ] into appropriate file names in the current directory,
5. substitution 3: interpretation of accent quoted strings, ` `, as commands and their execution,

[12] **Grouping**

- special argument `--`,
- commands may be grouped into brackets:
  - round brackets, (commands-sequence;) to group process which are to be run as a separate sub-process; may be run in background (&),
  - curly brackets, { commands-sequence; } just to group commands,
- command end recognized with: `<NL>` `;` `&`

[13] **Input/ output Redirection**

After session opening user environment contains the following elements:

- standard input (**stdin**) - stream 0,
- standard output (**stdout**) - stream 1,
- standardo error output (**stderr**) - stream 2.

There are the following redirection operators:

`> file`	redirect stdout to *file*
`>> file`	append stdout to *file*
`< file`	redirect stdin from *file*
`<< EOT`	read input stream directly from the following lines, till EOT word occurence.
`n > file`	redirect output stream with descriptor n to *file*,
`n >> file`	append output stream with descriptor n to *file*,
`n>&m`	redirect output of stream n to input of stream m,
`n<&m`	redirect input of stream n to output of stream m.

Commands grouped together in a common text file may be executed by:

```
$ sh [options] file_with_commands [arg ...]
```

After giving to the file execute permision by command: **chmod**, np.:

```
$ chmod +x plik_z_cmd
```

one can submit it as a command without giving **sh** before the text file name.

```
$ file_with_commands arg ...
```

[15] Compound Commands

- for steering of the shell script execution there are the following instructions: **if**, **for**, **while**, **until**, **case**
- it is possible to write **if** in a shorter way:

  And-if `&&` (when result equal to 0)
  Or-if `||` (when result different to 0)

  ```
  $ cp x y   &&   vi y
  $ cp x y   ||   cp z y
  ```

- Each command execution places in `$?` variable result of execution. The value ”0” means that the execution was succesful. Nonzero result means occurence of some error during command execution.

[16] ’if’ Instruction

- the standard structure of the compound

```
if if_list
        then then_list
        [ elif elif_list; then then_list ] ...
        [ else else_list ]
fi
```

- the **if_list** is executed. If its exit status is zero, the **then_list** is executed. Otherwise, each **elif_list** is executed in turn, and if its exit status is zero, the corresponding **then_list** is executed and the command completes. Otherwise, the **else_list** is executed, if present.

-
```
if cc -c p.c
then
  ld p.o
else
  echo "compilation error" 1>&2
fi
```

[17] **'case' Instruction**

- the standard structure of the compound

```
case word in
  pattern1)  list1;;
  pattern2)  list2;;
  *) list_default;;
esac
```

- a **case** command first expands **word**, and tries to match it against each **pattern** in turn, using the same matching rules as for path-name expansion.

- an example

```
case $# in
  0) echo 'usage: man name' 1>&2; exit 2;;
```

[18] **Loop Instructions**

In the **sh** command interpreter there are three types of loop instructions:

- ```
  for name [ in word ] ; do list ; done

  while list; do list; done
  until list; do list; done
  ```

- **for** instruction, executed once for each element of the for_list,
- **while** instruction, with loop executed while the condition returns 0 exit code (while condition is fulfilled),
- **until** instruction, with loop executed until the condition finally returns 0 exit code (loop executed while condition is not fulfilled),
- instructions **continue** and **break** may be used inside loops

```
#!/bin/sh
for i in /tmp /usr/tmp
do
        rm -rf $i/*
done
```

[19] **Different examples**

- ```
  $ cat file.dat | while read x y z
    do
      echo $x $y $z
    done
  ```

- ```
  #!/bin/sh
  i=1
  while [ $i -le 5 ]; do
    echo $i
    i=`expr $i + 1`
  done
  ```

- ```
  $ who -r
  . ru-level 2 Aug 21 16:58 2 0 S
  $ set `who -r`
  $ echo $6
  16:58
  ```

[20] **The Real-world Example**

```
#!/usr/bin/zsh
PATH=/usr/bin:/usr/local/bin:/bin
WAIT_TIME=5
. /export/home/oracle/.zshenv
# check whether it makes sense to check it
PID=`ps -ef | grep LISTENER | grep -v grep | awk -e '{print $2 }'`
if test -z "$PID"
then
        exit 0
fi
# check how it works
lsnrctl status >/dev/null 2>&1 &
sleep $WAIT_TIME
kill $! 2>/dev/null
res="$?"
if test "$res" != "1"
then
        kill $PID
        kill -9 $PID
        logger -p user.err Oracle LISTENER ERROR (stunned) - restarted
        lsnrctl start
fi
```