



SHELL PROGRAMMING

REFER SPA HAND-OUTS

SPA-SHELL PROGRAMMING

10-12-1999

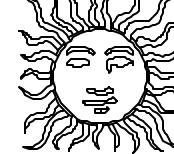
1



Dynamics of the Shell

- Built-in commands
- Scripts
- Variables
 - Environment
 - Local
- Wildcards
- Redirection
- Pipes
- Background processing
- Command substitution
- Sub-shells
- Sequences
 - Conditional
 - Unconditional

Shell Functions



SPA-SHELL PROGRAMMING

10-12-1999

2



Shell Preliminaries...

- **What is a Shell ?**

When you log on, a shell process is created by the Kernel to act upon your commands.

- **What are types of Shell?**

The default shell is the Bourne shell (sh).

If you don't like it, grab another shell.

C shell (csh)

Korn shell (ksh)

SPA-SHELL PROGRAMMING

10-12-1999

3



Shell Preliminaries...

- **What does the shell process do?**

It displays a prompt and waits for a user command

The shell executes a user command when entered and continues until an eof character (^D) is entered. The shell terminates upon reading eof character.


- **What if the command on line is too long ?**

If a command is longer than a line, terminate the line with a \ followed by **ENTER** and continue the command in the next line.

SPA-SHELL PROGRAMMING

10-12-1999


4



Special Characters

- > output redirection
- >> output redirection by appending
- < input redirection
- * file substitution wild card;
matches zero or more characters
- ? file subst. wild card; matches any single char
- [...] file subst wild card; matches any char in bracket
- `command` Command substitution;
replaced by the output from command
- | pipe symbol

SPA-SHELL PROGRAMMING
10-12-1999
5




Special Characters (cont..)

- ; Command sequencer
- || Conditional execution; executes next if previous failed
- && Conditional execution; executes next if previous succeeded
- (....) Group commands
- & Runs a command in the background
- # All characters that follow up to a new line are comment
- \$ Accesses a variable

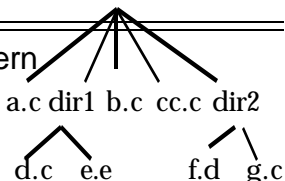
To turn off the special meaning of a shell metacharacter precede it with a \ or enclose it in single quotes ' '

SPA-SHELL PROGRAMMING
10-12-1999
6



File name substitution

The shell replaces the input pattern containing wildcards with matching file names




Examples

```
$ ls *.c
a.c b.c cc.c
$ ls ?.c
a.c b.c
$ ls [ac]*
a.c cc.c
```

```
$ ls [A-Za-z]*
a.c b.c cc.c dir1 dir2
$ ls dir*/*.c
dir1/d.c dir2/g.c
```

SPA-SHELL PROGRAMMING
10-12-1999
7



Command Substitution

Command substitution

```
$ echo there are `ls | wc -w` files in my directory
there are 5 files in my directory
```

Sequences

```
$ date;pwd;ls
Thu Jun 4 18:57:57 MAL 1998
/users/local/hitman/unix/myfiles
a.c b.c cc.c dir1 dir2
```

SPA-SHELL PROGRAMMING
10-12-1999
8



Conditional Sequencing

```
$ cc myprog.c && a.out
    Execute a.out only if cc was successful
$ cc myprog.c | | echo compilation failed
    Execute echo only if cc returns error
Every Unix process terminates with an exit value
    0      means success
    Nonzero (usually 1) means failure

exit 7 #sets value of exit status of 7
```

SPA-SHELL PROGRAMMING

10-12-1999

9



Background Processing

For a command, shell creates process to execute that command and sleeps until execution finishes.
If command is followed by **&**, shell returns to accept next command and previous command is run in background

```
$ find . -name d.c -print &
[1]      1429
$ date
Thu Jun  4 19:15:45 MAL 1998
$ ./unix/myfiles/dir1/d.c
```

SPA-SHELL PROGRAMMING

10-12-1999

10



Shell scripts

```
#!/bin/sh
echo "The date today is \c" # \c is to prevent newline
date
```

numFile

```
$ chmod 700 numfile
$ numfile
The date today is Thu Jun  4 19:26:45 MAL 1998
$
```

SPA-SHELL PROGRAMMING

10-12-1999

11



Sub-shells

Subshells are created when

- > You give a command to create a shell
\$ **/bin/sh**
- > Or, you give a group command
\$ **(ls; pwd; date)**
- > You give a script to execute
- > You give a background job to execute

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

SPA-SHELL PROGRAMMING

10-12-1999

12



Variables

Two kinds of variables > local and environment

Predefined environment variables

\$HOME full pathname of your home directory

\$PATH list of directories to search for commands

\$MAIL full pathname of your mail box

\$USER your user id

\$SHELL full pathname of your login shell

\$TERM type of your terminal

\$ echo home = \$HOME

home = /users/local/hitman

SPA-SHELL PROGRAMMING

10-12-1999

13



User-Created Variables

- Any sequence of **alphanumeric data**, beginning only with an **alphabet** can be used as a variable name.
- When a value is assigned to a variable, do not embed the ' = ' with a space or tab. Use " " if needed.
- Remove variables by assigning null value or use **unset** command
- Use **readonly** command to retain the previous value of a variable.
- Use **export** command for making local variables of the parent within the "grabs" of child processes.

SPA-SHELL PROGRAMMING

10-12-1999

14



User-Created Variables

Examples:

```
$ economy=excellent
```

```
$ echo economy
```

```
economy
```

```
$ echo $economy
```

```
excellent
```

```
$ echo "$economy"
```

```
excellent
```

```
$ echo '$economy'
```

```
$economy
```

```
$ echo \economy
```

```
\economy
```

SPA-SHELL PROGRAMMING

10-12-1999

15




```
$
$ echo "$vision"
solid, certain and crystal clear
$ echo $vision
solid, certain and crystal clear
$ vision="solid, certain and crystal clear"
$ echo "$vision"
solid, certain and crystal clear
$ echo '$vision'
$vision
$ echo $vision
solid, certain and crystal clear
$ readonly vision
$ vision="can we be less optimistic, please?"
sh: vision: This variable is read only.
$
```

SPA-SHELL PROGRAMMING

10-12-1999

16




Call by Value “grab” by child process

```
$ cat boom1
hat=small
echo "boom1 1: $hat"
boom
echo "boom1 2: $hat"
$ cat boom
echo "boom 1: $hat"
hat=big
echo "boom 2: $hat"
$ boom1
boom1 1: small
boom 1:
boom 2: big
boom1 2: small
```

```
$ cat boom2
export hat
hat=small
echo "boom2 1: $hat"
boom
echo "boom2 2: $hat"

$ boom2
boom2 1: small
boom 1: small
boom 2: big
boom2 2: small
```

SPA-SHELL PROGRAMMING
10-12-1999
17



Built-in Variables

Built-in variables with special meaning


- `$$` The process id of the shell
- `$0` The name of the shell script (if applicable)
- `$1...$9` `$n` refers to the nth command line argument
- `$*or $@` A list of all the command line arguments

```
$ sortCount data1 sfile poetry
14 62 400 total
4 12 67 data1
4 26 121 poetry
6 24 212 sfile
```

sortCount

wc \$* | sort

SPA-SHELL PROGRAMMING
10-12-1999
18




Built-in variables

Built-in variables with special meaning

- `$#` Number of arguments on the command line
- `$!` Value of PID number of last process (background)
- `$?` Exit status of the last “executed” process

SPA-SHELL PROGRAMMING
10-12-1999
19



Quoting

Single quotes (`'`) prevent

- wildcard replacement
- variable substitution command substitution

Double quotes (`"`) prevent wildcard replacement

```
$ echo 3 * 4 = 12
3 a.c b.c cc.c dir1 dir2 4 = 12
$ echo `3 * 4 = 12`
3 * 4 = 12
$ echo "3 * 4 = 12"
3 * 4 = 12
```

SPA-SHELL PROGRAMMING
10-12-1999
20



Example of Quotes

```
$ echo ` $MAIL knows today is `date` `
$MAIL knows today is `date`

$ echo "$MAIL knows today is `date`"
/var/mail/hitman knows today is Thu Jun 4
                                20:19:16 MAL 1998
With nested quotes, only the outer quotes have effect
```

SPA-SHELL PROGRAMMING

10-12-1999

21



Keyword Shell Variables

- HOME
- PATH
- MAIL
- Primary Prompt PS1
- Secondary Prompt PS2
- Absolute pathnames CDPATH
- Time Zone TZ

Check your **.profile** file to see the values of these variables.

SPA-SHELL PROGRAMMING

10-12-1999

22



Keyword Shell Variables (Examples)

```
$ PS1="`hostname`: "
apiitg30: echo test
test
apiitg30:PS1="$ "
$ echo "demo of prompt string
> 2"
demo of prompt string
2
$ PS2="secondary prompt: "
$ echo "this demonstrates
secondary prompt: prompt string 2"
this demonstrates
prompt string 2
$
```

SPA-SHELL PROGRAMMING

10-12-1999

23