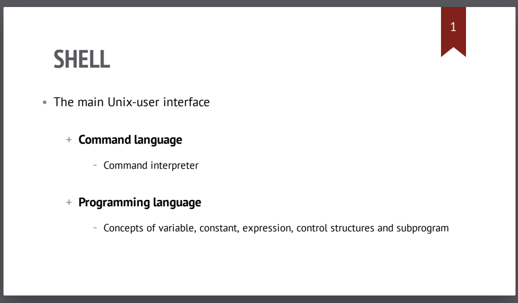
Operating systems 1, Lecture 2

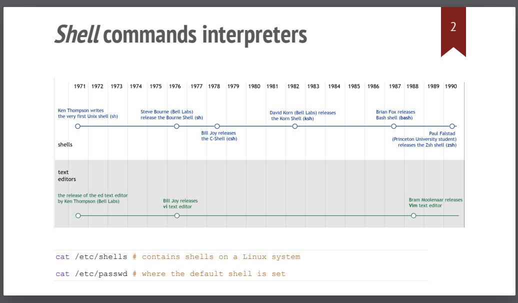
Sanda-Maria AVRAM, Ph. D.

Leave a feedback



SHELL

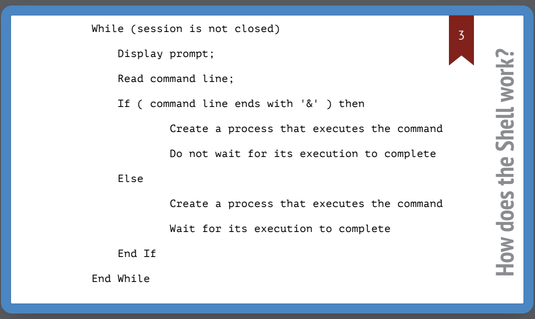
- The main Unix-user interface
  - **Command language**
    - Command interpreter
  - **Programming language**
    - Concepts of variable, constant, expression, control structures and subprogram



Shell commands interpreters

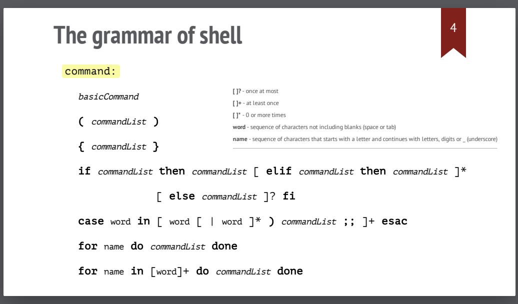
1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990
shells
Ken Thompson writes the very first Unix shell (sh)
Steve Bourne (Bell Labs) release the Bourne Shell (sh)
Bill Joy releases the C-Shell (csh)
David Korn (Bell Labs) releases the Korn Shell (ksh)
Brian Fox releases Bash shell (bash)
Paul Falstad (Princeton University student) releases the Zsh shell (zsh)
text editors
the release of the ed text editor by Ken Thompson (Bell Labs)
Bill Joy releases vi text editor
Bram Moolenaar releases Vim text editor

```
cat /etc/shells # contains shells on a Linux system
cat /etc/passwd # where the default shell is set
```



How does the Shell work?

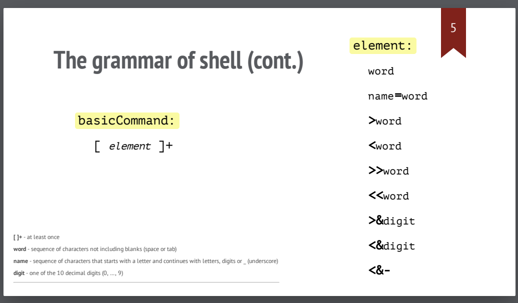
```
While (session is not closed)
    Display prompt;
    Read command line;
    If ( command line ends with '&' ) then
            Create a process that executes the command
            Do not wait for its execution to complete
    Else
            Create a process that executes the command
            Wait for its execution to complete
    End If
End While
```



The grammar of shell

```
command:
    basicCommand
    ( commandList )
    { commandList }
    if commandList then commandList [ elif commandList then commandList ]*
            [ else commandList ]? fi
    case word in [ word [ | word ]* ) commandList ;; ]+ esac
    for name do commandList done
    for name in [word]+ do commandList done
```

[ ]? - once at most
[ ]+ - at least once
[ ]* - 0 or more times
word - sequence of characters not including blanks (space or tab)
name - sequence of characters that starts with a letter and continues with letters, digits or _ (underscore)

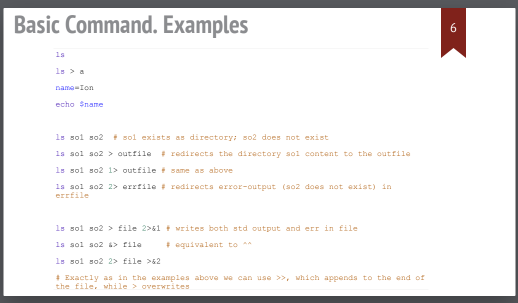


The grammar of shell (cont.)

```
basicCommand:
    [ element ]+
```

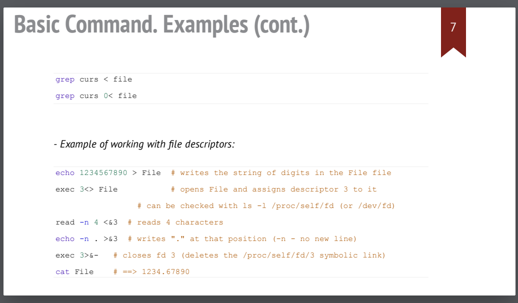
```
element:
    word
    name=word
    >word
    <word
    >>word
    <<word
    >&digit
    <&digit
    <&-
```

[ ]+ - at least once
word - sequence of characters not including blanks (space or tab)
name - sequence of characters that starts with a letter and continues with letters, digits or _ (underscore)
digit - one of the 10 decimal digits (0, ..., 9)



Basic Command. Examples

```
ls
ls > a
name=Ion
echo $name

ls so1 so2  # so1 exists as directory; so2 does not exist
ls so1 so2 > outfile  # redirects the directory so1 content to the outfile
ls so1 so2 1> outfile # same as above
ls so1 so2 2> errfile # redirects error-output (so2 does not exist) in
errfile

ls so1 so2 > file 2>&1 # writes both std output and err in file
ls so1 so2 &> file     # equivalent to ^^
ls so1 so2 2> file >&2
# Exactly as in the examples above we can use >>, which appends to the end of
the file, while > overwrites
```

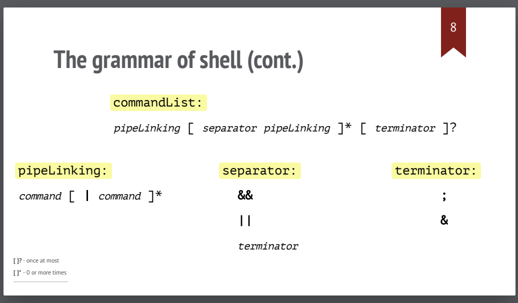


Basic Command. Examples (cont.)

```
grep curs < file
grep curs 0< file
```

- Example of working with file descriptors:

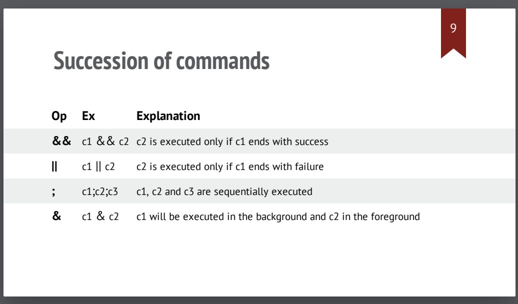
```
echo 1234567890 > File  # writes the string of digits in the File file
exec 3<> File           # opens File and assigns descriptor 3 to it
                 # can be checked with ls -l /proc/self/fd (or /dev/fd)
read -n 4 <&3  # reads 4 characters
echo -n . >&3  # writes "." at that position (-n - no new line)
exec 3>&-   # closes fd 3 (deletes the /proc/self/fd/3 symbolic link)
cat File    # ==> 1234.67890
```



The grammar of shell (cont.)

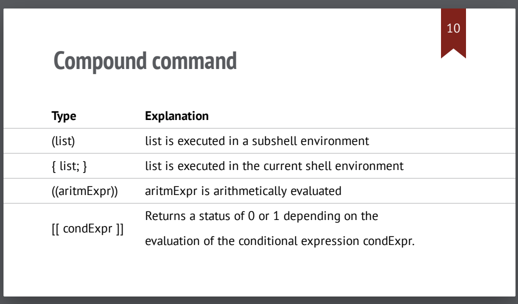
```
commandList:
    pipeLinking [ separator pipeLinking ]* [ terminator ]?

pipeLinking:                  separator:          terminator:
command [ | command ]*           &&                    ;
                                 ||                    &
                                 terminator
```

[ ]? - once at most
[ ]* - 0 or more times



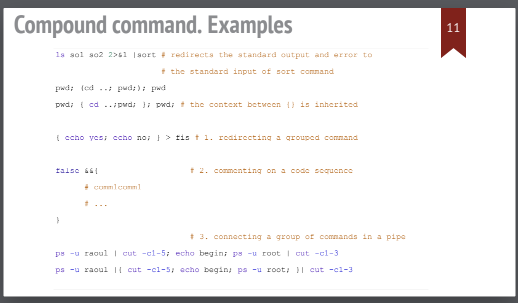
Succession of commands

Op	Ex	Explanation
&&	c1 && c2	c2 is executed only if c1 ends with success
||	c1 || c2	c2 is executed only if c1 ends with failure
;	c1;c2;c3	c1, c2 and c3 are sequentially executed
&	c1 & c2	c1 will be executed in the background and c2 in the foreground



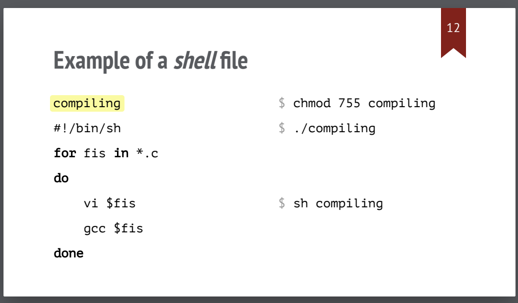
Compound command

Type	Explanation
(list)	list is executed in a subshell environment
{ list; }	list is executed in the current shell environment
((aritmExpr))	aritmExpr is arithmetically evaluated
[[condExpr]]	Returns a status of 0 or 1 depending on the evaluation of the conditional expression condExpr.



Compound command. Examples

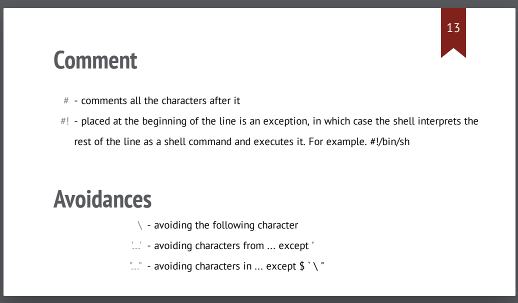
```
ls so1 so2 2>&1 |sort # redirects the standard output and error to
                      # the standard input of sort command
pwd; (cd ..; pwd;); pwd
pwd; { cd ..;pwd; }; pwd; # the context between {} is inherited

{ echo yes; echo no; } > fis # 1. redirecting a grouped command

false &&{                  # 2. commenting on a code sequence
      # comm1comm1
      # ...
}
                           # 3. connecting a group of commands in a pipe
ps -u raoul | cut -c1-5; echo begin; ps -u root | cut -c1-3
ps -u raoul |{ cut -c1-5; echo begin; ps -u root; }| cut -c1-3
```



Example of a *shell* file

```
compiling
#!/bin/sh
for fis in *.c
do
    vi $fis
    gcc $fis
done
```

```
$ chmod 755 compiling
$ ./compiling

$ sh compiling
```



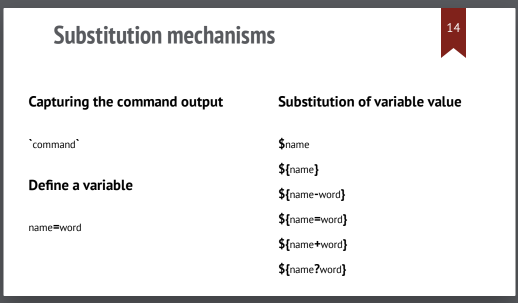
Comment

\# - comments all the characters after it

#! - placed at the beginning of the line is an exception, in which case the shell interprets the rest of the line as a shell command and executes it. For example. #!/bin/sh

Avoidances

\ - avoiding the following character
'...' - avoiding characters from ... except '
"..." - avoiding characters in ... except $ ` \ "



Substitution mechanisms

Capturing the command output

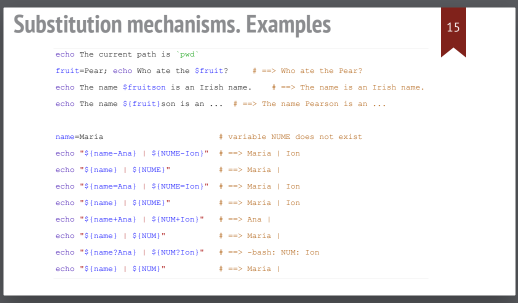
\`command\`

Define a variable

name=word

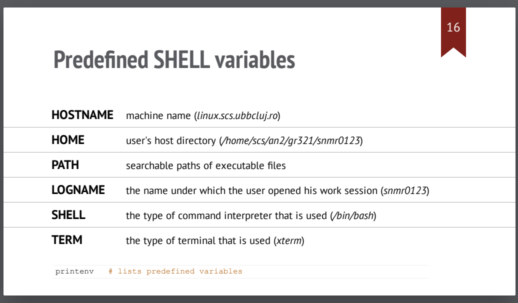
Substitution of variable value

$name
${name}
${name-word}
${name=word}
${name+word}
${name?word}



Substitution mechanisms. Examples

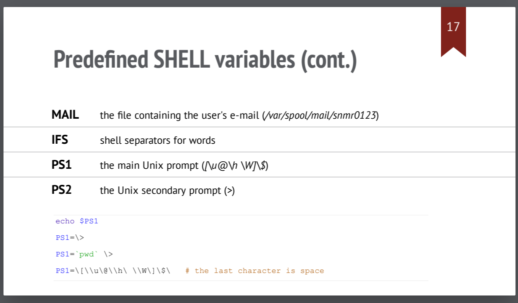
```
echo The current path is `pwd`
fruit=Pear; echo Who ate the $fruit?     # ==> Who ate the Pear?
echo The name $fruitson is an Irish name.    # ==> The name is an Irish name.
echo The name ${fruit}son is an ...  # ==> The name Pearson is an ...

name=Maria                         # variable NUME does not exist
echo "${name-Ana} | ${NUME-Ion}"  # ==> Maria | Ion
echo "${name} | ${NUME}"          # ==> Maria |
echo "${name=Ana} | ${NUME=Ion}"  # ==> Maria | Ion
echo "${name} | ${NUME}"          # ==> Maria | Ion
echo "${name+Ana} | ${NUM+Ion}"   # ==> Ana |
echo "${name} | ${NUM}"           # ==> Maria |
echo "${name?Ana} | ${NUM?Ion}"   # ==> -bash: NUM: Ion
echo "${name} | ${NUM}"           # ==> Maria |
```



Predefined SHELL variables

HOSTNAME	machine name (*linux.scs.ubbcluj.ro*)
HOME	user's host directory (*/home/scs/an2/gr321/snmr0123*)
PATH	searchable paths of executable files
LOGNAME	the name under which the user opened his work session (*snmr0123*)
SHELL	the type of command interpreter that is used (*/bin/bash*)
TERM	the type of terminal that is used (*xterm*)

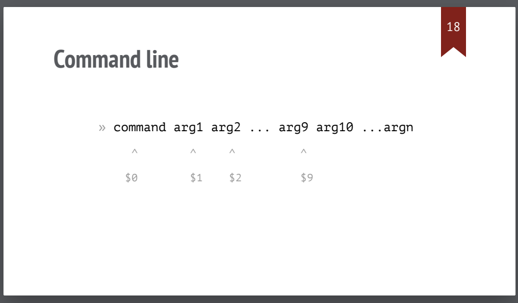
```
printenv   # lists predefined variables
```



Predefined SHELL variables (cont.)

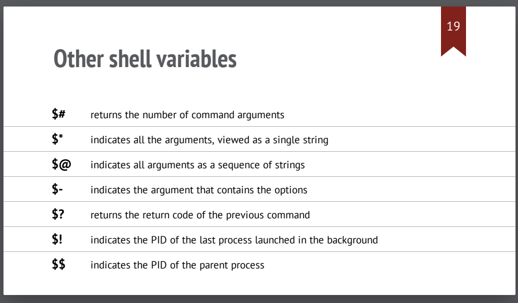
MAIL	the file containing the user's e-mail (*/var/spool/mail/snmr0123*)
IFS	shell separators for words
PS1	the main Unix prompt (*[\u@\h \W]\$*)
PS2	the Unix secondary prompt (>)

```
echo $PS1
PS1=\>
PS1=`pwd` \>
PS1=\[\\u\@\\h\ \\W\]\$\   # the last character is space
```



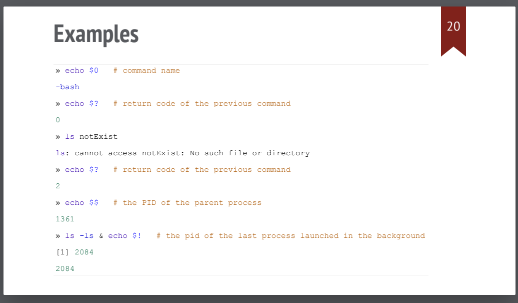
Command line

```
» command arg1 arg2 ... arg9 arg10 ...argn
     ^         ^    ^          ^
    $0        $1   $2         $9
```



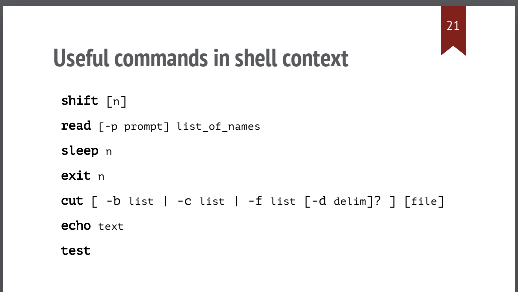
Other shell variables

$#	returns the number of command arguments
$*	indicates all the arguments, viewed as a single string
$@	indicates all arguments as a sequence of strings
$-	indicates the argument that contains the options
$?	returns the return code of the previous command
$!	indicates the PID of the last process launched in the background
$$	indicates the PID of the parent process



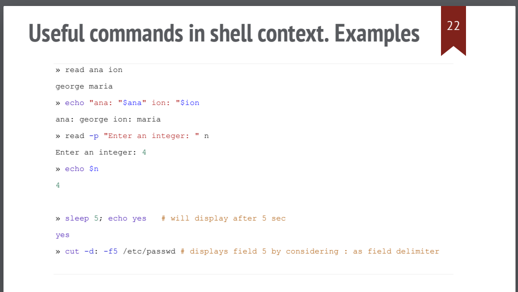
Examples

```
» echo $0   # command name
-bash
» echo $?   # return code of the previous command
0
» ls notExist
ls: cannot access notExist: No such file or directory
» echo $?   # return code of the previous command
2
» echo $$   # the PID of the parent process
1361
» ls -ls & echo $!   # the pid of the last process launched in the background
[1] 2084
2084
```



Useful commands in shell context

```
shift [n]
read [-p prompt] list_of_names
sleep n
exit n
cut [ -b list | -c list | -f list [-d delim]? ] [file]
echo text
test
```



Useful commands in shell context. Examples

```
» read ana ion
george maria
» echo "ana: "$ana" ion: "$ion
ana: george ion: maria
» read -p "Enter an integer: " n
Enter an integer: 4
» echo $n
4

» sleep 5; echo yes   # will display after 5 sec
yes
» cut -d: -f5 /etc/passwd # displays field 5 by considering : as field delimiter
```

test command

test *expr*

or

[*expr*]

Evaluates the expression value and returns exit status 0 or 1.

test command. Comparisons of integers

```
[ int1 -eq  int2 ]        # int1 = int2
[ int1 -ge  int2 ]        # int1 >= int2
[ int1 -gt  int2 ]        # int1 > int2
[ int1 -le  int2 ]        # int1 <= int2
[ int1 -lt  int2 ]        # int1 < int2
[ int1 -ne  int2 ]        # int1 != int2
```

test command. File comparisons

```
[ file1 -ot  file2 ]          # file1 older than file2
[ file1 -nt  file2 ]          # file1 newer than file2


    # considering the change time
```

test command. File tests

```
[ -d  file ]       # if file is a directory.
[ -e  file ]       # if file exists.
[ -f  file ]       # if file exists and is a regular file.
[ -L  file ]       # if file is a symbolic link.
[ -r  file ]       # if file is a readable file.
[ -w  file ]       # if file is a writeable file.
[ -x  file ]       # if file is an executable file.
```

test command. String comparisons

```
[ -z  string ]          # if string is empty string.
[ -n  string ]          # if string is non-empty string.

[ string1 =  string2 ]    # if string1 is the same as string2.
[ string1 !=  string2 ]   # if string1 is not the same as string2.
```

test command. Composing *test* expressions

```
     [ !E ]     # denial of expression E.
[ E1 -a  E2 ]   # AND composition of the E1 and E2 expressions.
[ E1 -o  E2 ]   # OR composition of the E1 and E2 expressions.
```

test command. Examples

```
» [ 3 -lt 4 -a 3 -gt 2 ]
» echo $?
0
» test -f infoCurs
» echo $?
0
» test -f info
» echo $?
1
» [ -d $HOME ]
» echo $?
0
» test -z $PATH
» echo $?
1
```

The End