**Positional and Special Parameters to the Shell**

**·Positional parameters are arguments passed to the Shell or script on the** **command line. Remember the syntax of the command line:**

command [option(s)] [argument(s)]

**· The command line delimiters are blanks and tabs.**

command param1 param2 

$0=command

$1=1st parameter

$2=2nd parameter

$#=number shell arguments

$$=process ID of current shell

$?=exit value of previous process

$-=options passed to shel1

$!=process ID of last background process

$\*,$@=positional parameters passed from command

line

**Notes**

· Only $1 through $9 can be used in a Shell script.

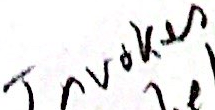
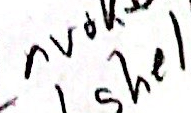
$ cat showarg

echo The 1st parameter is **$1**

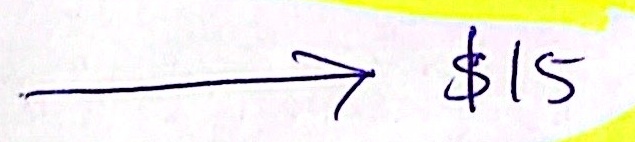
echo The 2nd parameter is **$2**

echo The command is $0

$ sh showarg hi ho here we go

**Positional Parameters to the Shell(Cont'd)**

·Theset **command can be used to force values into positional parameters.**

$ set `wc file`

$ echo $1 $2 $3

$ echo $3 $1 $2

1.2 3 y

$set a b"c d"e

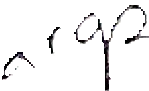
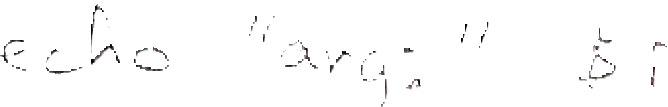
$ echo $#

$ echo $@

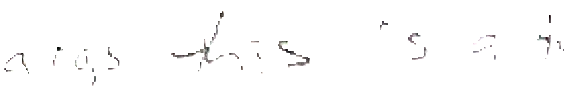
$ echo $\*→

**The parameters are treated as a single string.**

 $ for i in "$\*"

 > do > echo $i 

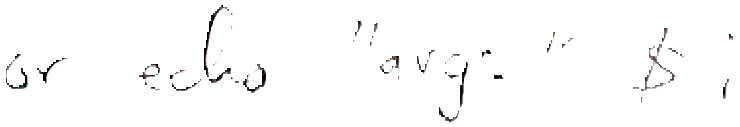
> done



**The parameters are treated as individual strings.**

$ for i in "$@"

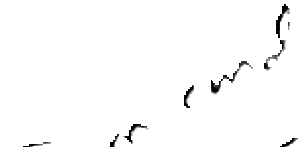
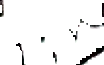
> do

> echo $i 

> done



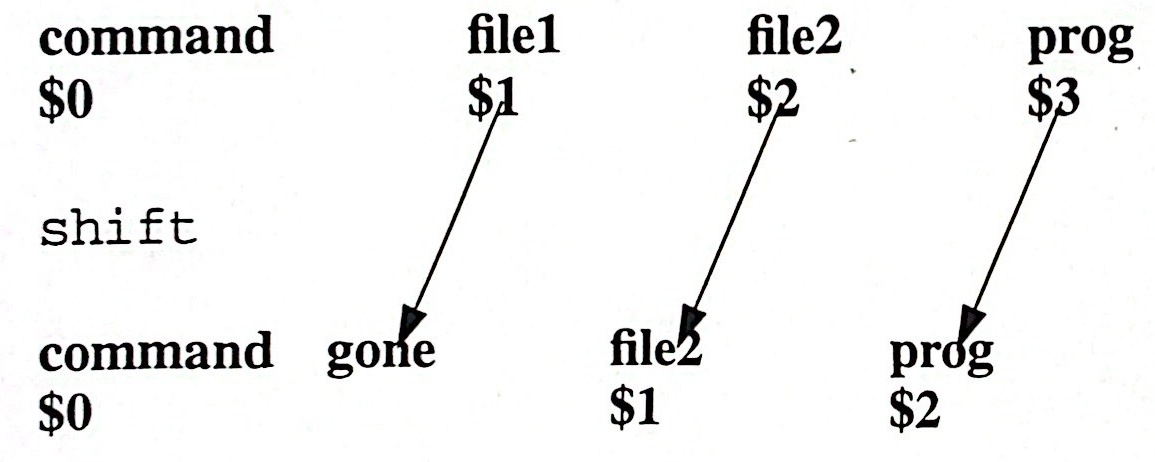
**Notes**

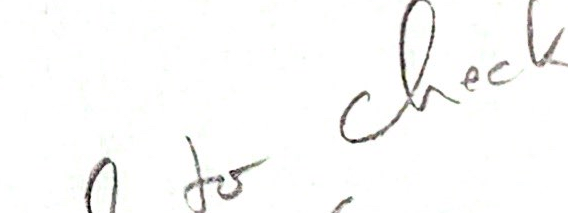
 

**Shifting Positional Parameters**

**·You** **can** shi ft the positional parameters **to** different positions.

**Example:**



**Notes**

·Notice that file1 is gone after the shift occurs.

· Inside a Shell script, you can assign the positional parameters to user defined variables to store them.

$a=$1

$ echo $a

**Positional Parameters to the Shell (Cont'd)**

**· Remember:**

**The command is stored in** $0.

**The number of arguments is stored in** $#.

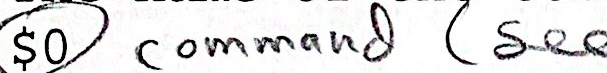
**· Exercise:**

**Create the file** name.arg.

$ cat name.arg

echo The Shell arguments are:

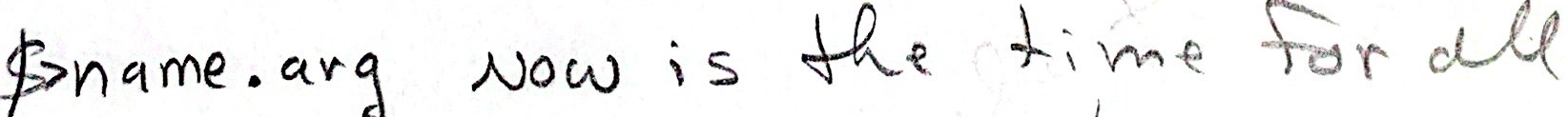
echo $1 $2 $3 $4 $5 $6 $7 $8 $9

echo The name of the command is:echo $O command 

echo The number of arguments is:

echo $#

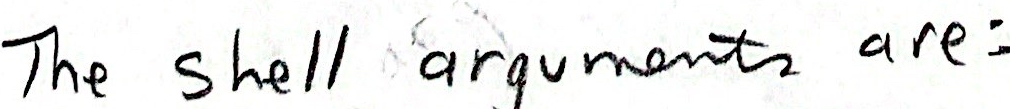
**Run name.arg,supplying your own arguments.**



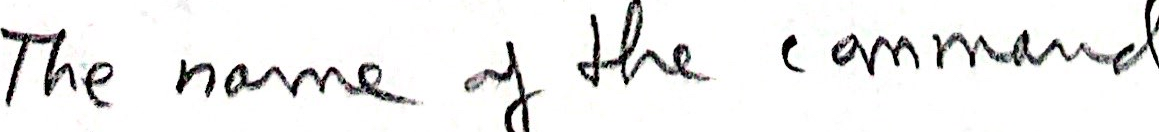


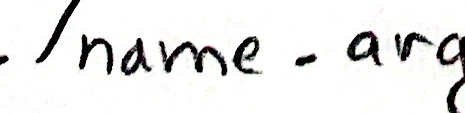
**Notes**

















Parameters

**Parameter Substitution**

·${parameter:-value}

It is possible to define a default value for the parameter. If the parameter is null,then value is substituted.

Example:

$ echo ${LPDEST:-1p}

**will** echo **the** **value** **of** LPDEST if it has been assignedand **1p otherwise.**

·${parameter:=value}

**If the parameter is null,** valueis

**substituted and also assigned to the**

**parameter. However it cannot be used to**

**set the default value of a positional**

**parameter.**

Example:

$ echo ${LPDEST:=1p}

**performs the same function** as${LPDEST:-1p}

**Legal** $ echo ${1:-a}

**Illegal** $ echo ${1:=a}

**Notes**

Parameters include the positional parameters, special Shell variables, and ordinary variables that are sometimes called keyword parameters. If a Shell parameter has not been assigned a value, it assumes the value of the null sting.

$ echo sa

or

$ echo ${a}

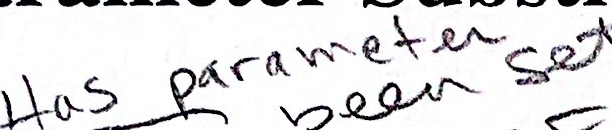
will echo nothing unless a has been assigned a value.

UNIX Shell Programming

6-7

© Copyright 1996-2000 GTRC,All Rights Reserved

**Parameter Substitution (Cont'd)**

 $ {parameter:?value}

**If the parameter has not been assigned a value and it is not appropriate to** **assign a default value, a message can be transmitted by using:**

**Example:**

$ echo ${LPDEST:?"What is your default printer?"}

${parameter:+yalue}

**If parameter is set and is not null, then** value **is used, otherwise no** **substitution is made.This is the opposite behavior of“:-".**

**Example:**

$ echo ${LPDEST:+1p}

**This overrides the choice of** LPDEST **if it is set.**

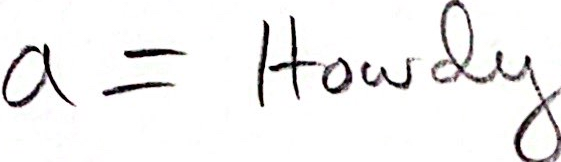


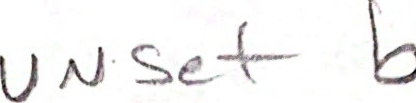
**Notes**

· If theparameter is null and no value is given after the ?, then the message “parameter null or not set” is displayed.

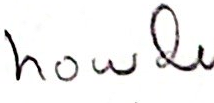
·The value after the + is not used unless it is to be used as the substitute string.

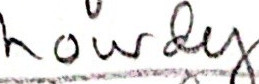
**Parameter Substitution Exercises**

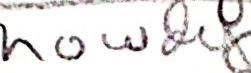
**· Assign a value to a.**



**·Execute the following commands.**

$ echo ${a} 

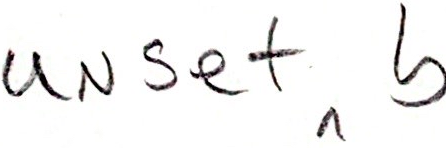
$ echo ${a:-surprise  

echo  

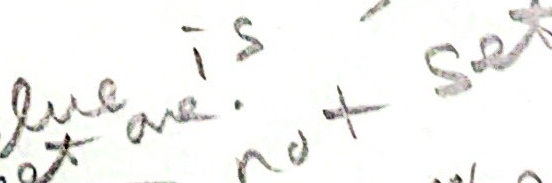
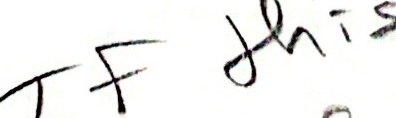
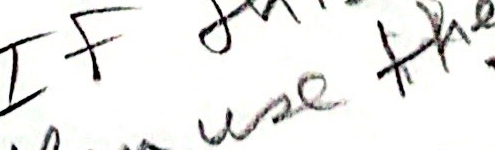
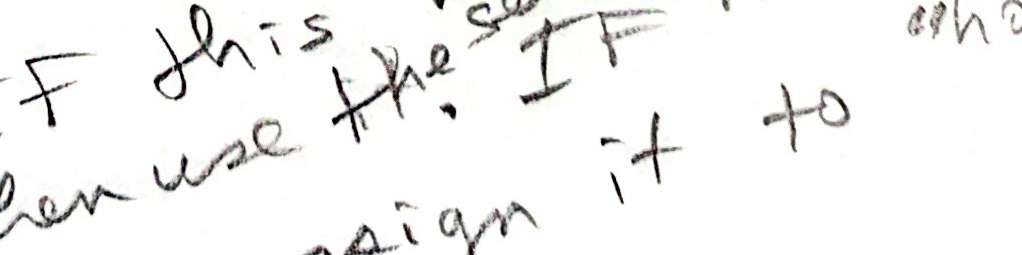
$echo?"you give a message'

**·Repeat the above commands replacing a with the parameter b where b has** **not been set.**



**Notes**



    :=  

**Parameter Substitution Summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Syntax | Parameter  Set | Parameter  NULL | Display | Parameter  Assignment |
| ${parameter | yes | no | orig-value | orig-value |
| ${parameter:-value} | yes | yes | value | orig-value |
| $ {parameter:=value} | yes | no | orig-value | orig-value |
| ${parameter | yes | yes | value | value |
| $ {parameter | yes | no | orig-value | orig-value |
| ${parameter:?value} | yes | yes | value | orig-value |
| ${parameter:+value} | yes | no | value | orig-value |
| $ {parameter:+value} | yes | yes | no subst | orig-value |



**Notes**

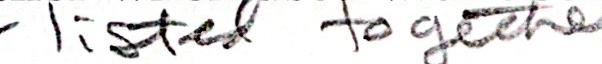
·There are similar parameter substitutions to the above, but without the colon. Their behavior is different when the parameter is set to NULL.

6-10

**Summary**

The arguments to a shell program are available in the positional parameters. 

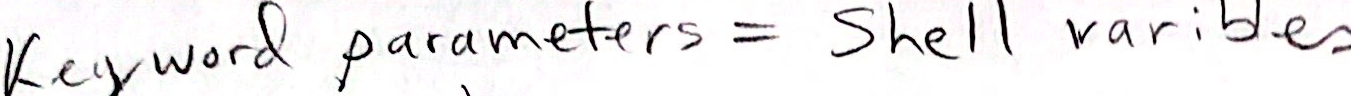
$1 through $9 are the first nine arguments. "$@" represents all the arguments. $\* is another notation that is often used. They are different from each other when used with double quotes.



Use shift to delete one or more positional parameters. The remaining parameters are renumbered starting at one.

There are a number of special parameter substitution syntaxes for using or assigning default values, printing error messages, or overriding the value of a variable.

Notes

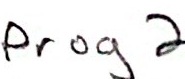


SAmE Parameters

**Lab 6**

**Objective:**

·There are two types of parameters, positional and keyword. If a parameter is a digit, it is a positional parameter. Positional parameters can also be assigned values using set. Keyword parameters can also be assigned by the following: name=value. In this lab you will investigate parameter name assignment.

   Prog 1 

**Exercises:**

1. Enter both of the following for programs:

A. for i in "$\* B. $ for i in "$@"

> do echo $i- > do echo $i

> done ≥done

$ $

Explain the difference in the two programs:

2. Take the output of the 'date' command and pass it to the set command



Use echo to display the values of the positional parameters < 

Also display the values associated with $#, $@, and $\*



Run the shi ft command and display the values aghows

3. Create a program that displays the name of the command and echo the values of the arguments listed on the command line.

Run the program and supply arguments on the command line (Hint: See Page 6-6)

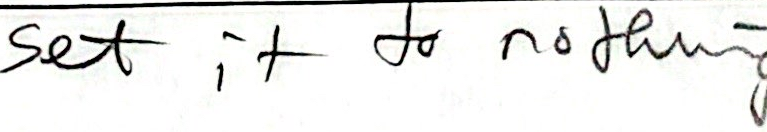
4. Use the preceding “Parameter Substitution Summary”. Assign a value to the parameter. echo each of the four different cases. Each time after using the “parameter substitution”,echo the value of the parameter. Notice in what case(s) the displayed value differs from the assigned value.

C

5. Now set the value of the parameter to NULL and repeat Exercise 4.

Example: Suppose your favorite editor is vi, but you do not have it defined.

|  |  |
| --- | --- |
| Command | Value (EDITOR) |
| $ EDITOR=  Secho ${EDITOR:-vi    vi  $ echo $EDITOR | nul1  nu11  nu11 |
| $ echo ${EDIT  vi  $ echo $EDITOR  vi | nu11  vi |
| EDITOR=  $ echo ${EDITOR:?"editor not assigned"}  $ echo $EDITOR | nu11  nu11 |
| EDITOR=vi  $ echo ${EDITOR:+emacs}  $ echo $EDITOR  vi | vi  emacs  vi |





**7**

**Pattern Matching and expr**

Objectives

- -Distinguish between Shell file **name** **pattern** matching and regular expression patterns

Construct and use Shell wild cards in pattern matching

Identify Shell metacharacters

Explain the effect of metacharacters on Shell special characters

Identify regular expression characters and their associated meaning

·Evaluate expressions using expr



Notes

**Shell File Name Substitution Characters**

\* **Matches zero or more characters**

? **Matches any single character**

[...] **Matches any one character in the set**

[123] **Matches 1, or 2,or** 3

[a-z] Matchesany **one character in the range a to z inclusive**

[^a-m] **Does not match any character between a and m**

**Notes**

The Shell provides the user a pattern matching mechanism. The above special characters are used in pattern matching by the Shell. They are not as sophisticated as those used by some of the utilities which will be covered later.

None of the metacharacters will match a “/' in a path name, e.g., "/usr\*dan" will not match "/usr/spool/mail/dan”.

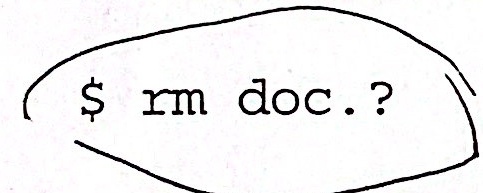
**Shell Pattern Matching Examples**

$ 1s \*

**Lists all files in the current directory excluding those that start with a dot.**

**Dots at the beginning of a file must be explicitly matched.**

$ 1s [a-m]\*

**Lists all files in the current directory that start with the letters a through** m.

**Removes all documents of the form,say** doc.a,doc.4,etc.

$ pr chap[1-5]

**Prints** **all files** chap1, chap2, chap3, chap4,and chap5.

$ rm /usr/\*/core

**Removes all core files in subdirectories of** /usr.



**Notes**

When a special character is enclosed in single quotes, it loses its special meaning. Also,if the special character is preceded by a\, it loses its special meaning.

**Quoting**

**·There are characters that have special meaning to the Shell. They are**:

、<> \* 1 & ? [ ] ; ( ) $

'\#^ RETURN Space Tab

**These characters are called metacharacters. They lose their special meaning** **if they are preceded by a backslash,** **\, or if they are enclosed in single** **quotes.**

$ echo \\*

\*

$ echo '\*\*\*

\*\*\*

**Notes**

The backslash is most efficiently used to quote a single character and the single quotes are used to quote a string of characters.



Pattern Matching and expr

**Metacharacters**

**Metacharacters:**

\ $ , "1

**Shell Special Characters:**

\* ? [ ]

**Evaluated**

**Terminator**

**Metacharacter**

**Terminator**

、

,

11

\$'"

"1

a11

**Example:**

$ set 'wc file'

$ echo "$\*"

$ echo "$@"

**Within double quotes, variables are expanded and command substitution** **occurs, but Shell wild cards are not expanded.**

**Notes**

Single quotes, double quotes, and grave quotes are used as terminators. The diagram displays the metacharacters that are evaluated between the given terminators.

·Inside double quotes parameter variable and command substitution occur.

UNIX Shell Programming

©Copyright 1996-2000 GTRC,All Rights Reserved

7-6

|  |  |  |  |
| --- | --- | --- | --- |
| Character | Meaning | Example | Matches |
|  | Any character | ch. | ch1,cha |
| < | Beginning of line | ^Tom | Tom Jones |
| $ | End of line | Jones$ | Tom Jones |
| ·$ | Single character at  end of line |  |  |
| ^$ | Blank line |  |  |
| \* | Zero or more occur-  rences of previous  regular expression | 1\*  11\* | Zero or more ones  One or more ones |
|  | Zero or more  characters |  | Entire line |
| t.\*e | t followed by zero or more characters  followed by e |  | the,these,te |
| [chars] | Any single charac-ter between [ ] | [tT] | t or T |
| [a-z] | Any character in  range |  |  |
|  | Any character not in range |  | Any character except a,b,and c. |
|  | Exactly <n>characters | [0-9]\{2\} | Exactly 2 integers |
|  | Between <min>and  <max> characters | [a-z]\{1,3\} | Between 1 and 3  letters |



Pattern Matching and expr

**Regular Expression Special Characters**

UNIX Shell Programming

7-7

©Copyright 1996-2000 GTRC, All Rights Reserved