CALL

Call one batch program from another, or call a subroutine.

```
Syntax
      CALL [drive:][path]filename [parameters]

      CALL :label [parameters]

      CALL internal_cmd

Key:
   pathname     The batch program to run.

   parameters   Any command-line arguments.

   :label       Jump to a label in the current batch script.

   internal_cmd Run an internal command, first expanding any variables in the argument.
```

The Microsoft help for the CALL command states: "*Calls one batch program from another without stopping the parent batch program*" to expand on this, it is true that the parent does not STOP, but it does PAUSE while the second script runs.

Examples

CALL a second batch file

The CALL command will launch a new batch file context along with any specified parameters. When the end of the second batch file is reached (or if EXIT is used), control will return to just after the initial CALL statement.

Arguments can be passed either as a simple string or using a variable:

```
CALL MyScript.cmd "1234"
CALL OtherScript.cmd %_MyVariable%
```

Example

```
::----------start main.cmd-----------
@Echo off
CALL function.cmd 10 first
Echo %_description% - %_number%

CALL function.cmd 15 second
Echo %_description% - %_number%
::----------end main.cmd-------------

::----------start function.cmd---------
@Echo off
:: Add 25 to %1
SET /a _number=%1 + 25
:: Store %2
SET _description=[%2]
::----------end function.cmd-----------
```

In many cases you will also want to use SETLOCAL and ENDLOCAL to keep variables in different batch files completely separate, this will avoid any potential problems if two scripts use the same variable name.

If you execute a second batch file *without* using CALL you may run into some buggy behaviour: if both batch files contain a label with the same name and you have previously used CALL to jump to that label in the first script, you will find execution of the second script starts at the same label. Even if the second label does not exist this will still raise an error *"cannot find the batch label"*. This bug can be avoided by always using CALL.

CALL a subroutine (:label)

The CALL command will pass control to the statement after the label specified along with any specified parameters.
To exit the subroutine specify `GOTO:eof` this will transfer control to the end of the current subroutine.

A label is defined by a single colon followed by a name:

```
:myShinyLabel
```

If placed inside brackets, the label must be followed by at least one valid command, even just an ECHO or REM command.
The maximum length for a label is 127 characters, no spaces.

This is the basis of a batch file function.

```
CALL :sub_display 123
CALL :sub_display 456
ECHO All Done
GOTO :eof

:sub_display
ECHO The result is %1
EXIT /B
```

At the end of the subroutine an `EXIT /B` will return to the position where you used CALL
(`GOTO :eof` can also be used for this)

Call and optionally continue

In some cases you may want to call a second batch file (or subroutine) and continue only if it succeeded.

```
CALL SecondScript.cmd || goto :eof
```

This will `goto:eof` if SecondScript.cmd returns an errorlevel greater than zero, you can force an errorlevel by using Exit /b 1

Passing by Reference

In addition to passing numeric or string values on the command line, it is also possible to pass a variable name and then use the variable to transfer data between scripts or subroutines. Passing by reference is a slightly more advanced technique but can be essential if the string contains characters that are CMD delimiters or quotes, otherwise passing a string like *Start & middle:"and & End* is likely to break something.

```
@Echo off
Echo:
Set "var1=Red Pippin"
Set "var2=St Edmunds Pippin"
Set "var3=Egremont Russet"

Echo: before: var1=%var1%  var2=%var2% var3=%var3%
call :myGetFunc var1 var2 var3
Echo: after: var1=%var1%  var2=%var2% var3=%var3%

Echo:&pause&goto:eof

::----------------------------------------------
::-- Function section starts below
::----------------------------------------------

:myGetFunc    - passing a variable by reference
Set "%~1=return64"
Set "%~3=return65"
EXIT /B
```

Buggy behaviour when using CALL

Redirection via a pipe '|' does not always work as expected.
This will work:
`CALL :function >file.txt`
This will fail:
`CALL :function | more`
More details can be found in this SO question: Why does delayed expansion fail when inside a piped block of code?

If the CALL command contains a caret character within a quoted string "test^ing" , the carets will be doubled.

Advanced usage - CALLing internal commands

CALL *command* [*command_parameters*]

CALL *command* can be used to run an executable or a second batch file, but can also be used to run any internal command (SET, ECHO etc) with the exception of FOR and IF. CALL will expand any variables passed on the same line.

`CALL REM` only partly works: redirection operators, conditional execution operators and brackets will be not remarked.

This is undocumented behaviour, in fact whenever CALL is run without a `:` prefix, it will always search disk for a batch file/executable called *command* before running the internal command. The effect of this extra disc access is that CALL SET is significantly slower than

CALL, its use in loops or with a large number of variables should be avoided.

Example

```
@Echo off
SETLOCAL
set _server=frodo
set _var=_server
CALL SET _result=%%%_var%%%
echo %_result%
```

The line shown in bold has the '%' symbols tripled, CALL will expand this to: `SET _result=frodo`

Each CALL does one substitution of the variables. (You can also do CALL CALL... for multiple substitutions)

In many cases, DelayedExpansion is a better/faster method:

```
@Echo Off
Setlocal EnableDelayedExpansion
Set _server=frodo
Set _var=_server
Set _result=!%_var%!
Echo %_result%
```

Errorlevels

If you run `CALL SET` this will reset `ERRORLEVEL = 0` even though normally SET ... will fail to reset an ERRORLEVEL
If you CALL a subroutine, the ERRORLEVEL will be left *unchanged*
If you CALL a subroutine, with a label that does not exist `ERRORLEVEL` will be set to 1
`(call )` will set the `ERRORLEVEL` to 0.
`(call)` will set the `ERRORLEVEL` to 1.

If you CALL an executable or resource kit utility make sure it's available on the machine where the batch will be running, test for it's existence with an IF command and throw an error if missing.

CALL is an internal command, (internally it is closely related to GOTO).
If Command Extensions are disabled, the CALL command will not accept batch labels.

"My mother never saw the irony in calling me a son-of-a-bitch" ~ Jack Nicholson

Related commands

How To: Functions - How to package blocks of code.
CMD - can be used to call a subsequent batch and ALWAYS return even if errors occur.
GOTO - jump to a label or GOTO :eof
ScriptRunner - Run one or more scripts in sequence.
START - Start a separate window to run a specified program or command.
Equivalent bash command (Linux): . (source) - Run a command script in the current shell, builtin - Run a shell builtin.