B

Special Variables

Default Variables and Parameters

Variable	Description

`$_`	This global scalar acts as a default variable for function arguments and pattern-searching space – with many common functions, if an argument is left unspecified, `$_` will be automatically assigned, so, for example, the following statements are equivalent:
	`chop($_)` and `chop`
	`$_ =~ m/`*`expr`*`/` and `m/`*`expr`*`/`

`@_`	The elements of this array are used to store function arguments, which can be accessed (from within a function definition) as `$_[`*`num`*`]`. The array is automatically local to each function.
`@ARGV`	The elements of this array contain the command line arguments intended for use by the script.
`$ARGV`	This contains the name of the current file when reading from the null filehandle `<>`. (`<>` is a literal, and defaults to standard input, `<STDIN>`, if no arguments are supplied from elements in `@ARGV`).

Regular Expression Variables (all read-only)

Variable	Description
`$(num)`	The scalar `$n` contains the substring matched to the *n*'th grouped subpattern in the last pattern match, and remains in scope until the next pattern match with subexpressions. It ignores matched patterns occurring in nested blocks that are already exited. If there are no corresponding groups, then the undefined value is returned.

`$&`	This scalar contains the string matched by the last successful pattern match. Once again, this won't include any strings matched in nested blocks. For example:
	`'UnicornNovember' =~ /Nov/;`
	`print $&;`
	will print Nov. For versions of perl since 5.005, this is not an expensive variable to use.

`$'`	This scalar holds the substring following whatever was matched by the last successful pattern match. For example, if we say: `'UnicornNovember' =~ /Nov/;` `$'` will return ember.
`$``	This scalar holds the substring preceding whatever was matched by the last successful pattern match. For example, if we say: `'UnicornNovember' =~ /Nov/;` `$`` will return Unicorn.
`$+`	This scalar holds the last substring matched to a grouped subpattern in the last search. It comes in handy if you're not sure which of a set of alternative subpatterns matched. For example, if you successfully match on `/(ab)*| (bc*)/`, then `$+` stores either `$1` or `$2`, depending on whether it was the first or second grouped subpattern that matched. For example, following: `'UnicornNovember' =~ /(Nov)\|(Dec)/;` `$+` will return Nov.

`@+`	This array lists the back pointer positions (in the referenced string) of the last successful match. The first element `@+[0]` contains the pointer's starting position following that match – each subsequent value corresponds to its position just *after* having matched the corresponding grouped subpattern. For example, following:
	`'UnicornNovember' =~ /(U)\w?(N)/;`
	`@+` will return (8,1,8), while following:
	`'UnicornNovember' =~ /(Uni)\w?(Nov)/;`
	`@+` will return (10,3,10).

Variable	Description

`@-`	This array lists the front pointer positions (in the referenced string) of the last successful match. The first element `@-[0]` contains the pointer's starting position prior to that match – each subsequent value corresponds to its position just *before* having matched the corresponding grouped subpattern. For example following:
	`'UnicornNovember' =~ /(Uni)\w?(Nov)/;`
	`@-` will return (0,0,7), while following:
	`'UnicornNovember' =~ /(Uni)(\w?)(Nov)/;`
	`@-` will return (0,0,3,7).

Input/Output Variables

Variable	Description
`$.`	This scalar holds the **current line number** of the last filehandle on which you performed either a `read`, `seek`, or `tell`. It is reset when the filehandle is closed. NB: `<>` never does an explicit close, so line numbers increase across `ARGV` files – also, localizing `$.` has the effect of also localizing perl's notion of 'the last read filehandle'.
`$/`	This scalar stores the **input record separator**, which by default is the newline `\n`. If it's set to "", input will be read one paragraph at a time.

`$\`	This scalar stores the **output record separator** for `print` – normally this will just output consecutive records without any separation (unless explicitly included). This variable allows you to set it for yourself. For example:
	`$\ = "-";`
	`print "one";`
	`print "two";`
	will print one-two-.

`$|`	This corresponds to an internal flag used by perl to determine whether buffering should be used on a program's write/read operations to/from files. If the value is `TRUE` (`$|` is greater than 0), buffering is disabled.

`$,`	This is the **output field separator** for `print` – normally this will just output consecutive fields without any separation (unless explicitly included). This variable allows you to set it for yourself. For example:
	`$, = "-";`
	`print "one","two";`
	will print one-two.

Table continued on following page

Variable	Description

`$"`	This is the **output field separator** for array values interpolated into a double-quoted string (or similar interpreted string) – the default is a space. For example:
	`$" = "-";`
	`@ar = ("one", "two", "three");`
	`print "@ar";`
	will print one-two-three.

Filehandle/format Variables

Variable	Description
`$#`	This holds the **output format** for printed numbers. **NB: The use of this variable has been deprecated.**
`$|`	This corresponds to an internal flag used by perl to determine whether **buffering** should be used on a program's write/read operations to/from files – if its value is `TRUE` (`$|` is greater than 0), then buffering is disabled.
`$%`	The current page number of the selected output channel.
`$=`	The current **page length**, measured in printable lines – the default is 60. This only becomes important when a top-of-page format is invoked – if a `write` command doesn't fit into a given number of lines, then the top-of-page format is used, before any printing past the page length continues.
`$-`	The number of lines left on a page – when a page is finished, it's given the value of `$=`, and is then decremented for each line outputted.
`$~`	The currently selected **format name** – the default is the name of the filehandle.
`$^`	The name of the **top-of-page format**.
`$:`	The set of characters after which a string may be broken to fill continuation fields (starting with `^`) in a format – default is `" \n-"` to break on whitespace or hyphens.
`$^L`	This holds a character that is used by a format's output to request a form feed – default is `\f`.

Error Variables

Variable	Description
`$?`	This holds the status value returned by the last pipe close, backtick (``` `` ```) command, or `system()` operator.
`$@`	This holds the **syntax error message** from the last `eval()` command – it evaluates to null if the last `eval()` parsed and executed correctly (although the operations you invoked may have failed in the normal fashion).

Variable	Description
`$!`	If used in a numeric context, this returns the current value of *errno*, with all the usual caveats. (so you shouldn't depend on `$!` to have any particular value unless you've got a specific error return indicating a system error.) If used in a string context, it returns the corresponding system error string. You can assign a set *errno* value to `$!` if, for instance, you want it to return the string for that error number, or you want to set the exit value for the `die()` operator.
`$^E`	This returns an **extended error message**, with information specific to the current operating system. At the moment, this only differs from `$!` under VMS, OS/2, and Win32 (and for MacPerl). On all other platforms, `$^E` is always the same as `$!`.

System Variables

Variable	Description
`$$`	The **process ID** (pid) of the Perl process running the current script.
`$<`	The **real user ID** (uid) of the current process.
`$>`	The **effective uid** of the current process. NB: `$<` and `$>` can only be swapped on machines supporting `setreuid()`.
`$(`	The **real group ID** (gid) of the current process.
`$)`	The **effective group ID** (gid) of the current process.
`$0` (zero)	The name of the file containing the Perl script being executed.
`$^X`	The name that the perl binary was executed as.
`$]`	The version number of the perl interpreter, including patchlevel / 1000 – can be used to determine whether the interpreter executing a script is within the right range of versions. See also `use VERSION` and `require VERSION` for a way to fail if the interpreter is too old.
`$^O`	The name of the operating system under which this copy of perl was built, as determined during the configuration process – identical to `$Config{'osname'}`.
`$^T`	The time at which the current script began running, in seconds since the beginning of 1970. Values returned by `-M`, `-A`, and `-C` filetests are based on this value.
`$^W`	The current value of the warning switch, either `TRUE` or `FALSE`.
`%ENV`	Your current environment – altering its value changes the environment for child processes.
`%SIG`	Used to set handlers for various signals.

Others

Variable	Description
`@INC`	A list of places to look for Perl scripts for evaluation by the `do EXPR`, `require`, or `use` constructs.
`%INC`	Contains entries for each filename that has been included via `do` or `require`. The key is the specified filename, and the value the location of the file actually found. The `require` command uses this array to determine whether a given file has already been included.