Perl Predefined Variables

Variable	Description	Example
$ARG **$_**	The default input and pattern-searching space. (Mnemonic: underline is understood in certain operations.)	`while (<>) {...} #equiv. only in while` `while (defined($_ = <>)) {...}`
$a **$b**	Special package variables when using sort(), see sort.	`@articles = sort {$a cmp $b} @files;`
$<digits>	Contains the sub-pattern from the corresponding set of capturing parentheses from the last pattern match. (Mnemonic: like \digits.)	
$MATCH **$&**	The string matched by the last successful pattern match. (Mnemonic: like & in some editors.)	`local $_ = 'abcdefghi';` `/def/;` `print "$` : $& : $'", "\n";` `# prints abc : def : ghi`
$PREMATCH **$`**	The string preceding whatever was matched by the last successful pattern match. (Mnemonic: ` often precedes a quoted string.)	
$POSTMATCH **$'**	The string following whatever was matched by the last successful pattern match. (Mnemonic: ' often follows a quoted string.)	

$LAST_PATTERN_MATCH
$+
	The text matched by the last bracket of the last successful search pattern.
	This is useful if you don't know which one of a set of alternative patterns matched. (Mnemonic: be positive and forward looking.)
		`/Version: (.*)|Revision: (.*)/`
		`&&`
		`($rev = $+);`

$^N	The text matched by the used group most-recently closed (i.e. the group with the rightmost closing parenthesis) of the last successful search pattern. (Mnemonic: the (possibly) Nested parenthesis that most recently closed.)	`$v = "sep:2:match";`
		`$v =~ /(?:(\d)(?{ $a = $^N }))/;`
		`print $a; # prints 2`

@LAST_MATCH_END **@+**	This array holds the offsets of the ends of the last successful submatches in the currently active dynamic scope.	**$+[0]** is the offset into the string of the end of the entire match. **$+[1]** is the offset past where $1 ends. You can use **$#+** to determine how many subgroups were in the last successful match.
$*	Set to a non-zero integer value to do multi-line matching within a string, 0 (or undefined) to tell Perl that it can assume that strings contain a single line, for the purpose of optimizing pattern matches. (Mnemonic: * matches multiple things.)	Use of **$*** is deprecated in modern Perl, supplanted by the /s and /m modifiers on pattern matching.
HANDLE->input_line_number(EXPR) **$INPUT_LINE_NUMBER** **$NR** **$.**	Current line number for the last filehandle accessed. (Mnemonic: many programs use "." to mean the current line number.)	
IO::Handle->input_record_separator(EXPR) **$INPUT_RECORD_SEPARATOR** **$RS** **$/**	The input record separator, newline by default. Setting **$/** to a reference to an integer, scalar containing an integer, or scalar that's convertible to an integer will attempt to read records instead of lines, with the maximum record size being the referenced integer. (Mnemonic: / delimits line boundaries when quoting poetry.)	`local $/;  # enable "slurp" mode` `local $_ = <FH>; # whole file now here`
HANDLE->autoflush(EXPR) **$OUTPUT_AUTOFLUSH** **$|**	If set to nonzero, forces a flush right away and after every write or print on the currently selected output channel. Default is 0. (Mnemonic: when you want your pipes to be piping hot.)	

IO::Handle->output_field_separator(EXPR)
$OUTPUT_FIELD_SEPARATOR
$OFS
$,
	The output field separator for the print operator. If defined, this value is printed between each of print's arguments. Default is undef. (Mnemonic: what is printed when there is a "," in your print statement.)	`@arr = (1,2,3);`
		`$, = " - "`
		`print @arr; # prints 1 – 2 - 3`

IO::Handle->output_record_separator(EXPR)
$OUTPUT_RECORD_SEPARATOR
$ORS
$
	The output record separator for the print operator. Default is undef.
	(Mnemonic: you set **$** instead of adding "\n" at the end of the print.)
		`@arr = (1, 2, "baz");`
		`$\ = "\t"`
		`foreach (@arr) { print }`
		`# prints 1 [tab] 2 [tab] baz`

$LIST_SEPARATOR
$"
	This is like **$,** except that it applies to array and slice values interpolated into a double-quoted string (or similar interpreted string). Default is a space.	`@arr = (“foo”, “esr”, “rms”);`
		`$” = “ - ”`
		`print “@arr”; # prints foo – esr – rms`

$SUBSCRIPT_SEPARATOR
$SUBSEP
$;
	The subscript separator for multidimensional array emulation. Default is "\034", the same as SUBSEP in **awk**. (Mnemonic: comma (the syntactic subscript separator) is a semi-semicolon.)	`If you refer to a hash element as`
		`$foo{$a,$b,$c} it really means`
		`$foo{join($;, $a, $b, $c)}`

$#	The output format for printed numbers. This variable is a half-hearted attempt to emulate **awk**'s OFMT variable. The initial value is "%.*n*g", where *n* is the value of the macro DBL_DIG from your system's *float.h*. (Mnemonic: # is the number sign.)	
HANDLE-> **format_page_number** **(EXPR)** **$FORMAT_PAGE_NUMBER** **$%**	The current page number of the currently selected output channel. Used with formats. (Mnemonic: % is page number in **nroff**.)	
HANDLE-> **format_lines_per_page** **(EXPR)** **$FORMAT_LINES_PER_PAGE** **$=**	The current page length (printable lines) of the currently selected output channel. Default is 60. Used with formats. (Mnemonic: = has horizontal lines.)	
HANDLE-> **format_lines_left(EXPR)** **$FORMAT_LINES_LEFT** **$-**	The number of lines left on the page of the currently selected output channel. Used with formats. (Mnemonic: lines_on_page - lines_printed.)	
@LAST_MATCH_START **@-**	**$-[0]** is the offset of the start of the last successful match. **$-[*n*]** is the offset of the start of the substring matched by *n*-th subpattern, or undef if the subpattern did not match.	`$` is same as substr($var, 0, $-[0])` `$& is the same as substr($var, $-[0], $+[0] - $-[0])` `$' is the same as substr($var, $+[0])` `$1 is the same as substr($var, $-[1], $+[1] - $-[1])` `$2 is the same as substr($var, $-[2], $+[2] - $-[2])` `$3 is the same as substr($var, $-[3], $+[3] - $-[3])`
HANDLE-> **format_name(EXPR)** **$FORMAT_NAME** **$~**	The name of the current report format for the currently selected output channel. Default is the name of the filehandle. (Mnemonic: brother to $^ .)	
HANDLE-> **format_top_name(EXPR)** **$FORMAT_TOP_NAME** **$^**	The name of the current top-of-page format for the currently selected output channel. Default is the name of the filehandle with _TOP appended. (Mnemonic: points to top of page.)	
IO::Handle-> **format_line_break** **_characters(EXPR)** **$FORMAT_LINE_BREAK** **_CHARACTERS** **$:**	The current set of characters after which a string may be broken to fill continuation fields (starting with ^) in a format. Default is " \n-", to break on whitespace or hyphens. (Mnemonic: a "colon" in poetry is a part of a line.)	
IO::Handle-> **format_formfeed(EXPR)** **$FORMAT_FORMFEED** **$^L**	What formats output as a form feed. Default is \f.	
$ACCUMULATOR **$^A**	The current value of the write() accumulator for format() lines. A format contains formline() calls that put their result into **$^A** . After calling its format, write() prints out the contents of **$^A** and empties. So you never really see the contents of **$^A** unless you call formline() yourself and then look at it.	
$CHILD_ERROR **$?**	The status returned by the last pipe close, backtick (``) command, successful call to wait() or waitpid(), or from the system() operator.	`The exit value of the subprocess is really ($?>>8), and $? & 127 gives which signal, if any, the process died from, and $? & 128 reports whether there was a core dump.`
${^ENCODING}	The *object reference* to the Encode object that is used to convert the source code to Unicode. Default is *undef*.	

$OS_ERROR
$ERRNO
$!
	If used numerically, yields the current value of the C errno variable, or in other words, if a system or library call fails, it sets this variable. (Mnemonic: What just went bang?)	`if (open(FH, $filename)) {`
		`  # Here $! is meaningless.`
		`  ...`
		`} else {`
		`  # ONLY here is $! meaningful.`
		`  ...`
		`  # Here $! might be meaningless.`
		`}`

%!	Each element of **%!** has a true value only if **$!** is set to that value.	`For example, $!{ENOENT} is true if and only if the current value of $! is ENOENT ; that is, if the most recent error was "No such file or directory"`
$EXTENDED_OS_ERROR **$^E**	Error information specific to the current operating system. (Mnemonic: Extra error explanation.)	
$EVAL_ERROR **$@**	The Perl syntax error message from the last eval() operator. (Mnemonic: Where was the syntax error "at"?)	
$PROCESS_ID **$PID** **$$**	The process number of the Perl running this script. (Mnemonic: same as shells.)	
$REAL_USER_ID **$UID** **$<**	The real uid of this process. (Mnemonic: it's the uid you came *from*, if you're running setuid.)	

$EFFECTIVE_USER_ID
$EUID
$>
	The effective uid of this process. (Mnemonic: it's the uid you went *to*, if you're running setuid.)	`$< = $>; # set real to effective uid`
		
		`# swap real and effective uid`
		`($<,$>) = ($>,$<);`

$REAL_GROUP_ID **$GID** **$(**	The real gid of this process. If you are on a machine that supports membership in multiple groups simultaneously, gives a space separated list of groups you are in. (Mnemonic: parentheses are used to *group* things. The real gid is the group you *left*, if you're running setgid.)	`The first number is the one returned by getgid(), and the subsequent ones by getgroups(), one of which may be the same as the first number.`
$EFFECTIVE_GROUP_ID **$EGID** **$)**	The effective gid of this process. If you are on a machine that supports membership in multiple groups simultaneously, gives a space separated list of groups you are in. (Mnemonic: parentheses are used to *group* things. The effective gid is the group that's *right* for you, if you're running setgid.)	`$) = "5 5"`
$PROGRAM_NAME **$0**	Contains the name of the program being executed. (Mnemonic: same as **sh** and **ksh**.)	
$[	The index of the first element in an array, and of the first character in a substring. Default is 0. (Mnemonic: [begins subscripts.)	
$]	The version + patchlevel / 1000 of the Perl interpreter. (Mnemonic: Is this version of perl in the right bracket?)	
$COMPILING **$^C**	The current value of the flag associated with the **-c** switch.	
$DEBUGGING **$^D**	The current value of the debugging flags. (Mnemonic: value of **-D** switch.)	
$SYSTEM_FD_MAX **$^F**	The maximum system file descriptor, ordinarily 2.	
$^H	This variable contains compile-time hints for the Perl interpreter.	`WARNING: This variable is strictly for internal use only. Its availability, behavior, and contents are subject to change without notice.`
%^H	The **%^H** hash provides the same scoping semantic as **$^H**. This makes it useful for implementation of lexically scoped pragmas.	`WARNING: This variable is strictly for internal use only. Its availability, behavior, and contents are subject to change without notice.`
$INPLACE_EDIT **$^I**	The current value of the inplace-edit extension. Use undef to disable inplace editing. (Mnemonic: value of **-i** switch.)	

$^M	By default, running out of memory is an untrappable, fatal error. However, if suitably built, Perl can use the contents of **$^M** as an emergency memory pool after die()ing.	`# allocate a 64K buffer for use in`
		`# an emergency if Perl was compiled`
		`# with -DPERL_EMERGENCY_SBRK`
		`$^M = 'a' x (1 << 16);`

$OSNAME **$^O**	The name of the operating system under which this copy of Perl was built, as determined during the configuration process.	
${^OPEN}	An internal variable used by PerlIO. A string in two parts, separated by a \0 byte, 1st part describes input layers, 2nd part descrabe output layers.	

$PERLDB **$^P**	The internal variable for debugging support.	
$LAST_REGEXP_CODE_RESULT **$^R**	The result of evaluation of the last successful (?{ code }) regular expression assertion (see perlre). May be written to.	

$EXCEPTIONS_BEING_CAUGHT
$^S
	Current state of the interpreter.	`$^S  State`
		`---------  ------------------`
		`undef  Parsing module/eval`
		`true (1)  Executing an eval`
		`false (0)  Otherwise`

$BASETIME **$^T**	The time at which the program began running, in seconds since the epoch (beginning of 1970). The values returned by the **-M**, **-A**, and **-C** filetests are based on this value.	
${^TAINT}	Reflects if taint mode is on or off. 1 for on (the program was run with **-T**), 0 for off, -1 when only taint warnings are enabled (i.e. with **-t** or **-TU**).	
${^UNICODE}	Reflects certain Unicode settings of Perl.	
${^UTF8LOCALE}	This variable indicates whether an UTF-8 locale was detected by perl at startup.	
$PERL_VERSION **$^V**	The revision, version, and subversion of the Perl interpreter, represented as a string composed of characters with those ordinals.	
$WARNING **$^W**	The current value of the warning switch, initially true if **-w** was used, false otherwise, but directly modifiable. (Mnemonic: related to the **-w** switch.)	
${^WARNING_BITS}	The current set of warning checks enabled by the use warnings pragma.	
$EXECUTABLE_NAME **$^X**	The name used to execute the current copy of Perl, from C's argv[0] or (where supported) */proc/self/exe*.	
ARGV	The special filehandle that iterates over command-line filenames in @ARGV. Usually written as the null filehandle in the angle operator <>	
$ARGV	Contains the name of the current file when reading from <>.	
@ARGV	The array **@ARGV** contains the command-line arguments intended for the script. **$#ARGV** is generally the number of arguments minus one, because **$ARGV[0]** is the first argument, *not* the program's command name itself.	
ARGVOUT	The special filehandle that points to the currently open output file when doing edit-in-place processing with **-i**. Useful when you have to do a lot of inserting and don't want to keep modifying $_.	
@F	The array @F contains the fields of each line read in when autosplit mode is turned on. See perlrun for the **-a** switch.	
@INC	The array @INC contains the list of places that the do EXPR , require, or use constructs look for their library files. It initially consists of the arguments to any **-I** command-line switches, followed by the default Perl library.	
@_	Within a subroutine the array **@_** contains the parameters passed to that subroutine.	
%INC	The hash **%INC** contains entries for each filename included via the do, require, or use operators.	
%ENV **$ENV{expr}**	The hash **%ENV** contains your current environment. Setting a value in ENV changes the environment for any child processes you subsequently fork() off.	

%SIG
$SIG{expr}
	The hash **%SIG** contains signal handlers for signals.	`sub handler {`
		`# 1st argument is signal name`
		`  my($sig) = @_;`
		`  print "Caught a SIG$sig\n";`
		`  close(LOG);`
		`  exit(0);`
		`}`
		`$SIG{'INT'}  = \&handler;`
		`$SIG{'QUIT'} = \&handler;`
		`# restore default action`
		`$SIG{'INT'}  = 'DEFAULT';`
		`# ignore SIGQUIT`
		`$SIG{'QUIT'} = 'IGNORE';`