Summary: Bash Parameter Expansion

Extracted from the Bash man page.

Expression	**Description**
${parameter:-word}	*Use Default Values*. If parameter is unset or null, the expansion of word is substituted. Otherwise, the value of parameter is substituted.
${parameter:=word}	*Assign Default Values*. If parameter is unset or null, the expansion of word is assigned to parameter. The value of parameter is then substituted. Positional parameters and special parameters may not be assigned to in this way.
${parameter:?word}	*Display Error if Null or Unset*. If parameter is null or unset, the expansion of word (or a message to that effect if word is not present) is written to the standard error and the shell, if it is not interactive, exits. Otherwise, the value of parameter is substituted.
${parameter:+word}	*Use Alternate Value*. If parameter is null or unset, nothing is substituted, otherwise the expansion of word is substituted.
${parameter:offset} ${parameter:offset:length}	*Substring Expansion*. Expands to up to length characters of parameter starting at the character specified by offset. If length is omitted, expands to the substring of parameter starting at the character specified by offset. length and offset are arithmetic expressions (see ARITHMETIC EVALUATION below). • If offset evaluates to a number less than zero, the value is used as an offset from the end of the value of parameter. Arithmetic expressions starting with a - must be separated by whitespace from the preceding : to be distinguished from the *Use Default Values* expansion. • If length evaluates to a number less than zero, and parameter is not @ and not an indexed or associative array, it is interpreted as an offset from the end of the value of parameter rather than a number of characters, and the expansion is the characters between the two offsets. • If parameter is @, the result is length positional parameters beginning at offset. • If parameter is an indexed array name subscripted by @ or *, the result is the length members of the array beginning with ${parameter[offset]}. A negative offset is taken relative to one greater than the maximum index of the specified array. • Substring expansion applied to an associative array produces undefined results. • Note that a negative offset must be separated from the colon by at least one space to avoid being confused with the ***:-*** expansion. • Substring indexing is zero-based unless the positional parameters are used, in which case the indexing starts at 1 by default. If offset is 0, and the positional parameters are used, $0 is prefixed to the list.
${!prefix*} ${!prefix@}	*Names matching prefix*. Expands to the names of variables whose names begin with prefix, separated by the first character of the IFS special variable. When @ is used and the expansion appears within double quotes, each variable name expands to a separate word.
${!name[@]}	*List of array keys*. If name is an array variable, expands to the list of array indices (keys) assigned in name. If

Expression	Description
${!name[*]}	name is not an array, expands to 0 if name is set and null otherwise. When @ is used and the expansion appears within double quotes, each key expands to a separate word.
${#parameter}	*Parameter length*. The length in characters of the value of parameter is substituted. If parameter is * or @, the value substituted is the number of positional parameters. If parameter is an array name subscripted by * or @, the value substituted is the number of elements in the array.
${parameter#word} ${parameter##word}	*Remove matching prefix pattern*. The word is expanded to produce a pattern just as in pathname expansion. If the pattern matches the beginning of the value of parameter, then the result of the expansion is the expanded value of parameter with the shortest matching pattern (the "#" case) or the longest matching pattern (the “##” case) deleted. If parameter is @ or *, the pattern removal operation is applied to each positional parameter in turn, and the expansion is the resultant list. If parameter is an array variable subscripted with @ or *, the pattern removal operation is applied to each member of the array in turn, and the expansion is the resultant list.
${parameter%word} ${parameter%%word}	*Remove matching suffix pattern*. The word is expanded to produce a pattern just as in pathname expansion. If the pattern matches a trailing portion of the expanded value of parameter, then the result of the expansion is the expanded value of parameter with the shortest matching pattern (the "%" case) or the longest matching pattern (the "%%" case) deleted. If parameter is @ or *, the pattern removal operation is applied to each positional parameter in turn, and the expansion is the resultant list. If parameter is an array variable subscripted with @ or *, the pattern removal operation is applied to each member of the array in turn, and the expansion is the resultant list.
${parameter/pattern/string} ${parameter//pattern/string}	*Pattern substitution*. The pattern is expanded to produce a pattern just as in pathname expansion. Parameter is expanded and the longest match of pattern against its value is replaced with string. If pattern begins with /, all matches of pattern are replaced with string. Normally only the first match is replaced. If pattern begins with #, it must match at the beginning of the expanded value of parameter. If pattern begins with %, it must match at the end of the expanded value of parameter. If string is null, matches of pattern are deleted and the / following pattern may be omitted. If parameter is @ or *, the substitution operation is applied to each positional parameter in turn, and the expansion is the resultant list. If parameter is an array variable subscripted with @ or *, the substitution operation is applied to each member of the array in turn, and the expansion is the resultant list.
${parameter^pattern} ${parameter^^pattern} ${parameter,pattern} ${parameter,,pattern}	*Case modification*. This expansion modifies the case of alphabetic characters in parameter. The pattern is expanded to produce a pattern just as in pathname expansion. The ^ operator converts lowercase letters matching pattern to uppercase; the , operator converts matching uppercase letters to lowercase. The ^^ and ,, expansions convert each matched character in the expanded value; the ^ and , expansions match and convert only the first character in the expanded value. If pattern is omitted, it is treated like a ?, which matches every character. If parameter is @ or *, the case modification operation is applied to each positional parameter in turn, and the expansion is the resultant list. If parameter is an array variable subscripted with @ or *, the case modification operation is applied to each member of the array in turn, and the expansion is the resultant list.