|  |  |
| --- | --- |
| Special Characters | |
| & | Background job |
| # | Comment |
| ~ | Home Directory |
| ! | Logical NOT |
| ‘ | Quote (Strong) |
| “ | Quote (Weak) |
| < | Redirect input |
| > | Redirect output |
| >> | Redirect output + append to file |
| | | Redirect (pipe) output to next command |
| / | Separator for pathname directories |
| ; | Separator for shell commands |
| [ ] | Start and end a character-set wildcard |
| { } | Start and end a command block |
| ( ) | Start and end a subshell |
| (( )) | Perform arithmetic |
| \* | Wildcard |
| ? | Wildcard – single character |
| $ | Variable expression |
| \ | Escape a special character |
| n>&m | Descriptor n is a copy of ouput file descriptor m |
| n<&m | Descriptor n is a copy of input file descriptor m |

|  |  |
| --- | --- |
| String Operators | |
| ${varname:-word} | Returns word |
| ${varname:=word} | Sets and returns word |
| ${varname:?message} | Prints message and exits |
| ${varname:offset:length} | Returns substring |
| ${varname:+word} | If varname is defined, return word |

|  |  |
| --- | --- |
| Pattern-matching operators | |
| ${varname#pattern} | Match first from the start |
| ${varname##pattern } | Match last from the start |
| ${varname%pattern} | Match first from the end |
| ${varname/pattern/replace} | Match longest and replace |
| ${varname//pattern/replace} | Match all and replace |

|  |  |
| --- | --- |
| Variables | |
| $0, $1, $2, | Positional parameters |
| $@ | “$1” “$2” “$3” … |
| $\* | A string of positional params > 0 |
| $# | Number of positional params |
| $? | Exit status of last command run |

|  |  |
| --- | --- |
| Functions | |
| define | function myfunction { … }  *or*  myfunction ( ) { … } |
| call | myfunction arg1 arg2 … |
| keywords | local – limit var scope |

|  |  |
| --- | --- |
| If / else conditions | |
| x && y | If x runs, then run y |
| x || y | If x fails, then run y |
| x -a y | x AND y |
| x -o y | x OR y |
| -lt, -le, -eq, -gt, -ge, -ne | Integer comparisons |
| =, !=, <, > | String comparisons |
| -n str1 | str1 has length > 0 (nonzero) |
| -z str1 | str1 has length 0 (zero) |
| -d file | File exists and is a directory |
| -e file | File exists |
| -f file | File exists and is a regular file |
| -r file | User has read permission on file |
| -s file | File exists and is non empty |
| -w file | User has write permission on file |
| -x file | User has execute permission on file, or search if directory |
| -N file | File was modified since it was left read |
| -O file | User owns file |
| -G file | File’s group ID matches the user’s group ID |
| file1 -nt file2 | file1 has newer modification time than file2 |

|  |  |
| --- | --- |
| Flow control sentences | |
| if | if *condition*; then  *commands*;  fi |
| for | for ((*init*; *condition*; *increment*)); do  *commands*;  done |
| for | for *var* in *array*; do  *commands*;  done |
| case | case *expression* in  *pattern1*) *commands* ;;  *pattern2*) *commands* ;;  \*) *commands* ;;  esac |
| while | while *condition*; do  *commands*;  done |
| until | until *condition*; do  *commands*;  done |

|  |  |
| --- | --- |
| Arrays | |
| Arr\_name=(‘el1’ ‘ el2’ ‘el3’) | define |
| Arr\_name[index] | Element #index |
| Arr\_name[-1] | Last element |
| Arr\_name[@] | All elements, space-separated |
| #Arr\_name[@] | Array length |
| #Arr\_name[index] | String length of the Nth element |
| Arr\_name[@]:m:n | Range (from position m, length n |
| !Arr\_name[@] | Keys of all elements |
| Arr\_name=("${Arr\_name[@]}” “newElement”) | Push |
| Arr\_name+=('newElement’) | Also Push |
| unset Arr\_name[n] | Remove one item |

|  |  |
| --- | --- |
| Dictionaries | |
| declare –A dict | Define |
| dict[key]=”value” | Define value of a key |
| dict[key] | Value of a key |
| dict[@a] | All values |
| !dict[@] | All keys |
| #dict[@] | Number of elements |
| Unset dict[key] | Delete the key |