KornShell (ksh) Cheatsheet

A quick reference guide for KornShell (ksh) scripting, covering essential syntax, commands, and features for efficient shell programming.

Ksh Basics & Syntax

Basic Syntax

#!/bin/ksh	Shebang line, specifies the interpreter for the script.
Variable Assignment	`variable=value` (No spaces around `=`) `readonly variable=value` (Makes the variable read-only)
Variable Usage	`$variable` or `${variable}` (for clarity, especially with concatenation)
Command Substitution	`$(command)` or `command`
Comments	`# This is a comment`
Exit Status	`$?` - Access the exit status of the last executed command (0 is success).
String Concatenation	`string="$var1$var2"`

Input/Output

Reading Input	`read variable` (Reads a line from standard input)
Printing Output	`print "message"` or `echo "message"` (Prints to standard output)
Redirecting Output	`command > file` (Redirect standard output to file, overwriting) `command >> file` (Append standard output to file) `command 2> file` (Redirect standard error to file) `command &> file` (Redirect both standard output and error to file)
Here Documents	`cat << EOF text... EOF` (Passes multi-line text to a command)

Variables

Types of Variables	String variables (default), integer variables (using `integer`), arrays (using `typeset -a`)
Exporting variables	`export variable` (makes the variable available to subprocesses)
Unsetting a variable	`unset variable`
Special variables	$0 (Script name), $1, $2, ... (Arguments), $# (Number of arguments), $$ (Process ID)

Control Flow

Conditional Statements (if/then/else)

```
if [ condition ]; then
  commands
elif [ condition ]; then
  commands
else
  commands
fi
```

Example:

```
if [ "$var" = "value" ]; then
  print "Var is value"
fi
```

Case Statements

```
case $variable in
  pattern1) commands ;;
  pattern2) commands ;;
  *) commands ;;
esac
```

Example:

```
case $option in
  -a) print "Option A" ;;
  -b) print "Option B" ;;
  *) print "Invalid option" ;;
esac
```

Loops

For Loop

```
for variable in list; do
  commands
done
```

Example: `for i in 1 2 3; do print $i; done`

While Loop

```
while [ condition ]; do
  commands
done
```

Example:

```
i=0
while [ $i -lt 5 ]; do
  print $i
  i=$((i+1))
done
```

Until Loop

```
until [ condition ]; do
  commands
done
```

Select Loop

```
select variable in list; do
  commands
  break #Important to add break
done
```

Functions and Arrays

Functions

Defining a Function	`function_name() { commands }`
	or
	`function function_name { commands }`

Calling a Function	`function_name arg1 arg2`
Returning Values	`return value` (Returns an exit status, 0-255. Use `print` for other values.)
Local Variables	`local variable=value` (Declares a variable with local scope)

Arrays

Declaring an Array	`typeset -a array_name`
Assigning Values	`array_name[0]=value1 array_name[1]=value2`
Accessing Elements	`echo ${array_name[0]}`
Array Length	`echo ${#array_name[*]}` or `echo ${#array_name[@]}`
All Elements	`echo ${array_name[*]}` or `echo ${array_name[@]}`

String Manipulation and Built-in Commands

String Manipulation

Substring Extraction	`echo ${variable:offset:length}`
String Length	`echo ${#variable}`
Pattern Replacement	`echo ${variable/pattern/replacement}` (Replaces first occurrence) `echo ${variable//pattern/replacement}` (Replaces all occurrences)
Case Conversion	`echo ${variable^^}` (Uppercase) `echo ${variable,,}` (Lowercase)

Built-in Commands

`pwd`	Print working directory
`cd directory`	Change directory
`ls`	List files and directories
`mkdir directory`	Create directory
`rm file`	Remove file
`cp source destination`	Copy file
`mv source destination`	Move or rename file
`test expression` or `[ expression ]`	Evaluate expression (used in conditionals)