KORN SHELL PROGRAMMING CHEAT SHEET

Special Characters

Metacharacters have special meaning to the shell unless quoted (by preceding it with a \ or enclosing it in **\` \`**)
Inside double quotes **" "** parameter and command substitution occur and \ quotes characters **\\\`"$**
Inside grave quotes **\` \`** then \ quotes characters **\\'$** and also **"** if grave quotes are within double quotes

Input / Output

Input and output of a command can be redirected:

<file	Use file as standard input (file descriptor 0)
>file	Use file as standard output (file descriptor 1)
>|file	As above, but overrides the noclobber option
>>file	Use file as standard output, appending if it already exists
<>file	open file for reading and writing as standard input
<<word	Here document. The shell script is read as standard input until word is encountered. Unless a character of word is quoted, parameter and command substitution occur. Newline is ignored. \ must be used to quote **\\ $ \`**
<<-word	As above, but leading tabs are stripped
<&digit	Standard input is duplicated from the file descriptor digit
>&digit	Standard output is duplicated from the file descriptor digit Eg. >&2 will redirect standard output to standard error
<&-	Close standard input
>&-	Close standard output

Commands

;	sequential execution, return status is that of the last in the list
&	background execution
(cd /tmp; ls; ls | wc -l) &	sequential background execution
&&	AND – execute both, return true of both succeed
||	OR execute the first, if false, execute the second, return true if either succeed
$(command)	stdout of command substituted eg. DATE=\`date +"%Y-%m-%d"\`
\`command\`	alternate form of above, but has problems handling quotes and backquotes

Functions

A name used as a simple command to call a compound command with new positional parameters.
fname() {command list}
Expansions occur during each execution of the function, not during the function definition.
Exit status of a function call is the exit status of the last command executed within the function.

Signals

The INT and QUIT signals are ignored for a command executing in the background while the monitor option is unset.
trap commands signals When a signal is received execute the commands (which could be a function name)
See /usr/include/sys/iso/signal_iso.h for list of signals

```
        end_program ()
        {
            rm $TMPFILE # delete temporary file if user types Ctrl-C
            exit 1
        }
        trap end_program HUP INT QUIT TERM
```

Options

Use + to turn these options back off.

set -A NAME arguments	Array assignment, assigning sequential values from arguments
set -a	All subsequently defined variables are exported automatically
set -C	noclobber. Prevents existing files from being overwritten by redirection
set -n	Read commands in the script without executing them
set -x	Prints commands and arguments as the are executed (debugging)

Execution

If a command name matches a built-in command, it is executed within the current shell process.
Otherwise, if a command name matches a user defined function, the function is executed within the current shell process.
Otherwise, a process is created and an attempt is made to execute the command using exec searching $PATH to find an executable file if the filespec does not begin with a /. If the file has the execute permission bit set, but the file is not an executable program, it is assumed to be a text file containing shell commands and a sub-shell is spawned to read it. The sub-shell does not include non-exported aliases, functions and variables. However, a parenthesized command is executed in a sub-shell that includes the current environment.

. file params	Read the complete file, then execute the commands within the current shell environment. $PATH is used if necessary to find the file.
alias -x name=value	Create an alias for a command. Eg. alias ll="ls -al" -x exports the alias to scripts invoked by name

```
for NAME in $LIST; do
     commands
done
```

Each time through the loop, the next word of LIST is assigned to NAME

```
while commands; do          while [ expression ]; do
     more commands
done
```

Loop as long as the last of the commands return a status of 0

```
until commands; do
     more commands
done
```

Loop as long as the last of the commands returns a status of non-zero

```
case $NAME in
     pattern) commands ;;
     pattern) commands ;;
     *) commands ;;          #default if no previous patterns matched
esac
```

(commands)	Execute commands in a separate environment
break	Exit from the enclosed for, while or until loop.
break #	Exit from # nested loops
continue	Start the next iteration of the enclosed for, while or until loop
continue #	Start the next iteration of the # nested for, while or until loop
return	Causes shell function or **.** script to return to the invoking script. The return status is that of the last executed command. Status value is least significant 8 bits. Works like exit if invoked while not in a function or **.** script.
return status	As above, but specifying the status
exit status	Causes the shell to exit with the specified status value
cd	Change current working directory to $HOME
cd -	Change current working directory to the previous one
cd directory	Change current directory to the specified one
cd old new	Substitute the new string for the old string in the current directory name and change directory to the result
echo	Prints arguments on standard output (see also the printf utility, not part of the shell)
exec arg	Command specified by the arguments is executed in the current process (replacing this shell)
pwd	Output the absolute pathname of the current working directory
read NAME1 NAME2 NAME3 ...	One line of standard input is read and broken up using the $IFS characters as separators. The fields are assigned to the NAMEs in order, except that leftover fields are all assigned to the last one.
readonly NAME	$NAME cannot be changed by subsequent assignment
umask value	Set the permission bits to be stripped when creating files and directories umask 077 is most secure

Environment Variables

Variables marked using export or typeset -x become part of the environment that is inherited by executed commands. The environment can be augmented by preceding a command with a variable assignment.

Eg.	VAR=value command arguments
	(export VAR; VAR=value; command arguments)
export name	Mark the variable for automatic export to subsequently executed commands
typeset attribs NAME=value ...	
	Sets attributes and assigned values to shell variables
	If invoked within a function defines a new local instance of the variable
	Attributes: (**+** turns off the attributes)

-i	value is an integer. This makes arithmetic faster.
-l	converts uppercase to lowercase
-u	converts lowercase to uppercase
-r	read only
-x	automatic export to environment of subsequent commands
-H	UNIX to host file name mapping on non-UNIX systems
-L	left justifies and removes leading blanks from value
-L#	as above, defining the width of the field, right justifying with blanks or truncating
-R#	right justifies and fills with leading blanks, or truncates from the left
-Z#	right justifies and fills with leading zeros if the first non-blank character is a digit

Common Environment Variables

PS1	primary prompt string eg. "$"
PS2	secondary prompt string eg. ">"
ENV	pathname of script to execute when an interactive shell is started (like a dot script)
IFS	Input Field Separators – characters used for splitting fields. Default tab, space, newline
PATH	Directory search path for executables
PWD	Present working directory set by cd
TMPDIR	good place for temporary files

Filename Expansion

*	Matches any string, including null
?	Matches any single character
[...]	Matches any single character in this list [a-d] is the same as [abcd]
[!...]	Matches any single character not in this list

Positional Parameters

$0	The command itself
$1	First parameter $2 is 2nd, etc.
$*	All the parameters $1 $2 etc. If within double quotes a single word is generated with a space between each parameter
$@	All parameters $1 $2 etc. If within double quotes, each parameter expands to a separate word
$#	A decimal value which is the number of parameters (including the command parameter)
$?	The value of the exit status of the last executed foreground command. 0 is true.
$$	The process ID of the shell
$!	The process ID of the last background command
shift	Positional parameters are moved so $1=$2 $2=$3 etc
shift number	Positional parameters are shifted by the number specified (less than or equal to $#)

Named Variables

$NAME	
${NAME}	Equivalent, but needed if following characters are legal in as a name

If a named parameter is exported, it becomes an Environment Variable and is available to programs spawned.

Modification of Variables

${NAME:-word}	If NAME is unset or null, word is used instead
${NAME:=word}	If NAME is unset or null, word is assigned to NAME and used (does not work for $1 $2 etc.)
${NAME:?}	If NAME is unset or null an error message is sent to stderr

${NAME:+word}	If NAME is unset or null, the null string is used, otherwise word is used

If the colon is omitted from the above the test is only for NAME being unset

Variable Expansion

If expansion occurs within double quotes, pathname expansion and field splitting is not performed on the result.
suffix and prefix are subject to tilde expansion, parameter expansion, command substitution and arithmetic expansion.

${#NAME}	The number of characters in NAME
${NAME%suffix}	Strip the smallest suffix from NAME before using it (eg. remove filename extension)
${NAME%%suffix}	Strip the largest suffix from NAME before using it
${NAME#prefix}	Strip the smallest prefix from NAME before using it
${NAME##prefix}	Strip the largest prefix from NAME before using it
~logname/filepath	Substutes ~logname for the home directory ie. /export/home/logname/filepath If logname is omitted, the $HOME environment variable is used
${NAME[element]}	Use the value of an array variable. Element can be an arithmetic expression.

Arrays

Set -A NAME John David Smith *equivalent to* NAME[0]=John ; NAME[1]=David; NAME[2]=Smith
echo ${NAME[*]} *equivalent to* echo ${NAME[0]} ${NAME[1]} ${NAME[2]}

Conditional Expressions

Used to test file attributes and compare strings.

(expression)	true if expression is true. Used to group expressions.
! expression	true if expression is false
test expression	Evaluates conditional expressions - old Bourne syntax – use [[ ]] or (( ))
exp1 && exp2	true if both expressions are true
exp1 || exp2	true if either expression is true

((exp1 == exp2))	true if the expressions are equal	*Need spaces around brackets*
((exp1 != exp2))	true if the expressions are not equal	
((exp1 < exp2))	true if exp1 is less than exp2	
((exp1 <= exp2))	true if exp1 is less than or equal to exp2	
((exp1 > exp2))	true if exp1 is greater than exp2	
((exp1 >= exp2))	true if exp1 is greater than or equal to exp2	

[[string=pattern]]	true if the string matches pattern
[[string != pattern]]	true if the string does not match the pattern
[[string1 < string2]]	true if string1 sorts before string2 in locale
[[string1 > string2]]	true if string1 sorts after string2 in locale
[[-n string]]	true if length of string is greater than zero
[[-z string]]	true if length of string is zero
[[string]]	true if the string is not the null string

```
if commands; then
    commands
elif commands; then
    commands
else
    commands
fi
```

```
PREFERRED
if [ -w filename ]; then
    commands
elif [[ $VAR = "abc" ]] then
    commands
else
    commands
fi
```

```
ALTERNATIVE
if test -w filename; then
    commands
elif test "$VAR" = "abc"; then
    commands
else
    commands
fi
```

[-a file]	true if file exists
[-d file]	true if file is a directory
[-e file]	true if file exists
[-f file]	true if file is an ordinary file
[-r file]	true if file is readable by the current process
[-w file]	true if file is writable by the current process
[-x file]	true if file is executable by the current process (if a directory, has search permission)
[-s file]	true if file length is greater than zero
[file1 -nt file2]	true if file1 is newer than file2

[file1 -ot file2]	true if file1 is older than file2
[file1 -ef file2]	true if file1 and file2 refer to the same file
[-L file]	true if file is a symbolic link
[-p file]	true if the file is a pipe of fifo special file
[-b file]	true if file is a block special file
[-c file]	true if file is a character special file
[-S file]	true if file is a socket
[-O file]	true if file is owned by the effective user ID of the current process
[-G file]	true if the group of the file matches the effective group ID of the current process
[-u file]	true if the file has the set user ID permission bit set
[-g file]	true if the file has the group ID permission bit set
[-k file]	true if the file has the sticky permission bit set
[-t fildes]	true if the file descriptor is open and associated with a terminal device
[-o option]	true if option is turned on

Arithmetic Expressions

Expressions can be used when assigning an integer variable, as numeric arguments to ***test***, and with ***let*** to assign a value to a variable. Use () to override precedence.

	unary minus	==	equals
!	logical not	!-	not equals
*	multiply	<	less than
/	divide	<=	less than or equal
%	modulus	>	greater than
+	add	>=	greater than or equal
	subtract		

let A=B*C	assign A as the product of B times C
typeset -i A	create integer variable A

Arithmetic Evaluation

let performs integer arithmetic using long integers.
Constants may be in another base as base#value, so 16#20 is 0x20 which is decimal 32.
Precedence and associativity of operators are the same as C language.
Parameter substitution syntax is not used to reference variables.

Command Line Argument Processing

getopts optlist NAME
optlist is the string of command line option letters to be recognized (**-** or **+** can be used with options)
If a colon trails the letter, the option requires an argument.
The getopts command places the next option letter found in $NAME (**+** is prepended to the letter if specified)
The option's argument, if any, is stored in $OPTARG
Begin optlist with a colon to suppress shell error messages for unrecognized options (then handle errors in the script)

```
while getopts “:l:tv” OPT; do
    case “$OPT” in
    a)   LOGFILE=$OPTARG ;;
    t)   TESTFLAG=Y ;;
    +t)  TESTFLAG=N ;;
    v)   VERBOSE=Y ;;
    +v)  VERBOSE=N ;;
    ?)   echo “Invalid option $OPTARG”; exit 1 ;;
    esac
done
shift $OPTIND-1
echo There are $# remaining parameters which are $@
```

Shell Initialization

Note: Common to bourne shell initialization also, so commands must be compatible with both, or test $0 for the shell

/etc/profile	common to all users
$HOME/.profile	specific to each user
$ENV	run on each invocation of an interactive shell eg. ENV=$HOME/.kshrc