CHEATSHEET FOR

Bash scripting

Pattern substitution

```
STR=/path/to/foo.c

echo ${STR%.c}      #=> "/path/to/foo"
echo ${STR%.c}.o    #=> "/path/to/foo.o"
echo ${STR##*.}     #=> "c" (extension)

BASE=${STR##*/}     #=> "foo.c" (basepath)
DIR=${SRC%$BASE}    #=> "/path/to"
```

Substitutions by regex

```
echo ${STR/hi/hello}          # Replace first match
echo ${STR//hi/hello}         # Replace all matches
echo ${STR/#hi/hello}         # ^hi
echo ${STR/%hi/hello}         # hi$

echo "${STR:0:3}"             # .substr(0, 3) -- position, length
echo "${STR:-3:3}"            # Negative position = from the right

echo ${#line}                 # Length of $line

[ -z "$CC" ] && CC=gcc        # CC ||= "gcc"  assignment
${CC:=gcc}                    # $CC || "gcc"
${CC:-gcc}                    # same as above
```

Reading input

```
echo -n "Proceed? [y/n]: "
read ans
```

```
echo $ans

read -n 1 ans    # Just one character
```

Loops

Basic for loop

```
for i in /etc/rc.*; do
  echo $i
done
```

Ranges

```
for i in {1..5}; do
    echo "Welcome $i"
done
```

Reading lines

```
cat file.txt | while read line; do
  echo $line
done
```

Functions

Defining functions

```
myfunc() { ... }
fuction myfunc { ... }
fuction myfunc() { ... }
```

Returning strings

```
myfunc() {
    local myresult='some value'
    echo $myresult
}

result=$(myfunc)
```

Errors

```
myfunc() { return 1; }
```

Arguments

```
$#                      # Number of arguments
$*                      # All args
$1                      # First argument
```

Ifs - files

```
# File conditions
if [ -a FILE ]; then        # -e exists          -d directory        -f file
fi                          # -r readable        -w writeable        -x executable
                            # -h symlink         -s size > 0

# File comparisons
if [ FILE1 -nt FILE2 ]      # -nt   1 more recent than 2
                            # -ot   2 more recent than 1
                            # -ef   same files
```

Ifs

```
# String
if [ -z STRING ]            # empty?
if [ -n STRING ]            # not empty?

# Numeric
if [ $? -eq 0 ]             # -eq -ne -lt -le -gt -ge
                            # $? is exit status by the way

# Etc
if [ -o noclobber ]         # if OPTIONNAME is enabled
if [ ! EXPR ]               # not
if [ ONE -a TWO ]           # and
if [ ONE -o TWO ]           # or

# Regex
if [[ "A" =~ "." ]]
```

Numeric comparisons

```
if (( $a < $b ))
```

Unset variables

Assume $FOO is not set. Doing *this* will result in *that*:

```
${FOO:-word}              # Returns word
${FOO:+word}              # Returns empty, or word if set
${FOO:=word}              # Sets parameter to word, returns word
${FOO:?message}           # Echoes message and exits

${FOO=word}               # : is optional in all of the above
```

Numeric calculations

```
$((RANDOM%=200))          # Random number 0..200
$((a + 200))              # $ is optional
```

Arrays

```
# Declaring using declare -a
declare -a Fruits=('Apple' 'Banana' 'Orange')

Fruits[0]="Apple"
```

```
Fruits[1]="Banana"
Fruits[2]="Orange"

echo ${Fruits[0]}           # Element #0
echo ${Fruits[@]}           # All elements, space-separated
echo ${#Fruits[@]}          # Number of elements
echo ${#Fruits}             # String length of the 1st element
echo ${#Fruits[3]}          # String length of the Nth element
echo ${Fruits[@]:3:2}       # Range (from position 3, length 2)
```

Operations

```
Fruits=("${Fruits[@]}" "Watermelon")    # Push
Fruits=( ${Fruits[@]/Ap*/} )            # Remove by regex match
unset Fruits[2]                         # Remove one item
Fruits=("${Fruits[@]}")                 # Duplicate
Fruits=("${Fruits[@]}" "${Veggies[@]}") # Concatenate
lines=(`cat "logfile"`)                 # Read from file
```

Iteration

```
for i in "${arrayName[@]}"; do
  echo $i
done
```

Misc crap

```
command -V cd               #=> "cd is a function/alias/whatever"
```

Options

```
set -o noclobber            # Avoid overlay files (echo "hi" > foo)
set -o errexit              # Used to exit upon error, avoiding cascading errors
set -o pipefail             # Unveils hidden failures
set -o nounset              # Exposes unset variables
```

Glob options

```
set -o nullglob             # Non-matching globs are removed  ('*.foo' => '')
set -o failglob             # Non-matching globs throw errors
set -o nocaseglob           # Case insensitive globs
set -o globdots             # Wildcards match dotfiles ("*.sh" => ".foo.sh")
set -o globstar             # Allow ** for recursive matches ('lib/**/*.rb' => 'lib/a/b/c
```

set GLOBIGNORE as a colon-separated list of patterns to be removed from glob matches.

Trap errors

```
trap 'echo Error at about $LINENO' ERR
```

or

```
traperr() {
  echo "ERROR: ${BASH_SOURCE[1]} at about ${BASH_LINENO[0]}"
}

set -o errtrace
trap traperr ERR
```

Case/switch

```
case $1 in
  start | up)
    vagrant up
    ;;

  *)
    echo "Usage: $0 {start|stop|ssh}"
    ;;
esac
```

Source relative

```
source "${0%/*}/../share/foo.sh"
```

printf

```
printf "Hello %s, I'm %s" Sven Olga
```

Directory of script

```
DIR="${0%/*}"
```

Getting options

```
while [[ "$1" =~ ^- && ! "$1" == "--" ]]; do case $1 in
  -V | --version )
    echo $version
    exit
    ;;
  -s | --string )
    shift; string=$1
    ;;
```

```
  -f | --flag )
    flag=1
    ;;
esac; shift; done
if [[ "$1" == '--' ]]; then shift; fi
```

Heredoc

```
cat <<END
hello world
END
```

Reference

- Bash-hackers wiki (back-hackers.org)
- Shell vars (back-hackers.org)