Bash

re-run as other user

```
if [ $UID == 0 ]; then
    exec su -c "$0" john
fi
```

Check if command is installed

```
if ! command -v "${TOOL}" &>/dev/null; then
    echo " Command ${TOOL} not found. Please install it"
    exit 1
fi
```

and use it inside loop:

```
for tool in awk bc sed; do
    ....
done
```

Output

```
DEBUG() {
    if [ $DEBUG_ENABLE -eq 1 ]; then
    >&2 echo "DBG: $@"
    fi
}
```

Variables with space

```
# Treat arguments as parts of one argument
declare DST=$*

# Use double colons to pass argument with spaces
nice ionice -c idle btrfs filesystem defragment -v -r -czlib "${DST}"
```

Quote problematic characters to use in shell invocation:

```
  QUOTED_VAR="${VAR@Q}"
```

Default values

```
${parameter:-word}
    If parameter is unset or null, the expansion of word is substituted.
    Otherwise, the value of parameter is substituted.
```

```
FOO=${VARIABLE:-default}
```

Or, which will assign to VARIABLE as well:

```
FOO=${VARIABLE:=default}
```

reading in loop without subshell

```
var=0
while read file
do
  echo $file; var=1
done < <(ls -1 /tmp/)
```

'read' insid loop

In loop where stdin is redirected the easiest way to use read is:

```
TTY=`tty`
while read ...
do
  read CTRLC < ${TTY}
done < somedata
```

concatenate output (subshell)

```
( command1 ; command2 ; command3 ) | cat
{ command1 ; command2 ; command3 ; } > outfile.txt
```

leading zeros / octal

To interpret a number as decimal, use 10#n form, eg. 10#09

```
VAR=077; echo $(($VAR+1))
# 64
```

```
# specify base method
VAR=077; echo $((10#$VAR+1))
# 78

# strip leading zeros method
VAR=077; VAR=${VAR#0}; echo $(($VAR+1))
# 78
```

include/source other scripts

Possible test scenario:

1. call using from another directory full path /home/user/bin/myscripth.sh
2. add /home/user/bin to PATH and then call from another directory by “myscript.sh”
3. create symlink to script and call using symlink. Check for relative and absolute symlinks.
4. it should work also if it is sourced “source” or “.”

The best is to create setup/installer script which hardcode path to one shared script.

Another solutions:

```
#!/bin/sh
MY_DIR=$(dirname $(readlink -f $0))
$MY_DIR/other_script.sh
```

```
DIR=$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )
```

```
# Works correctly if script is sourced from another one
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && cd -P "$( dirname
"${BASH_SOURCE[0]}" )" && pwd )"
```

```
cur_file="${BASH_SOURCE[${#BASH_SOURCE[@]} - 1]}"
cur_dir="$(dirname "${cur_file}")"
```

```
`BASH_SOURCE' An array variable whose members are the source filenames where
the corresponding shell function names in the `FUNCNAME' array variable are
defined. The shell function `${FUNCNAME[$i]}' is defined in the file
`${BASH_SOURCE[$i]}' and called from `${BASH_SOURCE[$i+1]}'
```

double brackets [[ ]]

DBLBRACKETS

Bash double brackets Bash (extended test command) are safer and provides more features but they are not portable (and not sh compatible). Features:

1. not filename expansion. Strings will be take literally (i.e. “name*”).
2. operators like || instead of -o

3. regex matching =~

```
[[ -e $file_name ]]
```

```
[ -e "$file_name" ]
```

Check if string contains another string

not compatible with sh

```
if [[ $string == *"word1 word2"* ]] ...
```

```
if [[ $string =~ .*word.* ]] ...
```

Execute command until fail

```
while command; do :; done
```