

# bash

## Arbeiten mit Variablen

### Arrays

#### Array erstellen

```
# Array mit virtuellen Maschinen anlegen
machines=( vm01, vm02, vm03)

# Erstellt eine Liste mit den virtuellen Maschinen
IFS=';' read -r -a arrhosts <<< $(echo $machines | tr ',' ';')
```

#### Array iterieren

```
echo "Hier eine Liste der virtuellen Maschinen : "
i=0
for vm in "${arrhosts[@]}"
do
    echo $i - "$vm"
    i=$((i+1))
done

#Ausgabe:
0 - vm01
2 - vm02
3 - vm03
```

### Scripte mit Parametern

# Beispiel mit Abfrage eines bestimmten Wertes der Parametern

<https://stackoverflow.com/a/16496491>

```
#!/bin/bash

usage() { echo "Usage: $0 [-s <45| 90>] [-p <string>]" 1>&2; exit 1; }

while getopts ":s:p:" o; do
    case "${o}" in
        s)
            s=${OPTARG}
            (( s == 45 || s == 90 )) || usage
            ;;
        p)
            p=${OPTARG}
            ;;
        *)
            usage
            ;;
    esac
done
shift $(( OPTIND-1 ))

if [ -z "${s}" ] || [ -z "${p}" ]; then
    usage
fi

echo "s = ${s}"
echo "p = ${p}"
```

Ausgabe:

```
$ ./myscript.sh
Usage: ./myscript.sh [-s <45| 90>] [-p <string>]

$ ./myscript.sh -h
Usage: ./myscript.sh [-s <45| 90>] [-p <string>]
```

```
$ ./myscript.sh -s "" -p ""
Usage: ./myscript.sh [-s <45| 90>] [-p <string>]

$ ./myscript.sh -s 10 -p foo
Usage: ./myscript.sh [-s <45| 90>] [-p <string>]

$ ./myscript.sh -s 45 -p foo
s = 45
p = foo

$ ./myscript.sh -s 90 -p bar
s = 90
p = bar
```

## Beispiel mit getopt

<http://mywiki.woledge.org/BashFAQ/035>

```
#!/bin/sh
# Usage info
show_help() {
cat << EOF
Usage: ${0##*/} [-hv] [-f OUTFILE] [FILE]...
Do stuff with FILE and write the result to standard output. With no FILE
or when FILE is -, read standard input.

-h          display this help and exit
-f OUTFILE  write the result to OUTFILE instead of standard output.
-v          verbose mode. Can be used multiple times for increased
            verbosity.
EOF
}

# Initialize our own variables:
output_file=""
verbose=0

OPTIND=1
```

```

# Resetting OPTIND is necessary if getopt was used previously in the script.
# It is a good idea to make OPTIND local if you process options in a function.

while getopts hvf: opt; do
    case $opt in
        h)
            show_help
            exit 0
            ;;
        v)  verbose=$(( verbose+1))
            ;;
        f)  output_file=$OPTARG
            ;;
        *)
            show_help >&2
            exit 1
            ;;
    esac
done

shift "$((OPTIND-1))"  # Discard the options and sentinel --

# Everything that's left in "$@" is a non-option.  In our case, a FILE to process.
printf ' verbose=<%d>\noutput_file=<%s>\nLeftovers: \n' "$verbose" "$output_file"
printf ' <%s>\n' "$@"

# End of file

```

# Erstellen von Passwoertern

Quelle: <https://www.howtogeek.com/30184/10-ways-to-generate-a-random-password-from-the-command-line/>

## Generate a Random Password

For any of these random password commands, you can either modify them to output a different password length, or you can just use the first x characters of the generated password if you don't want such a long password. Hopefully you're using a password manager like [LastPass](#) anyway so

you don't need to memorize them.

This method uses SHA to hash the date, runs through base64, and then outputs the top 32 characters.

```
“ date +%s | sha256sum | base64 | head -c 32 ; echo
```

This method used the built-in /dev/urandom feature, and filters out only characters that you would normally use in a password. Then it outputs the top 32.

```
“ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c${1:-32}; echo;
```

This one uses openssl's rand function, which may not be installed on your system. Good thing there's lots of other examples, right?

```
“ openssl rand -base64 32
```

This one works a lot like the other urandom one, but just does the work in reverse. Bash is very powerful!

```
“ tr -cd '[:alnum:]' < /dev/urandom | fold -w30 | head -n1
```

Here's another example that filters using the strings command, which outputs printable strings from a file, which in this case is the urandom feature.

```
“ strings /dev/urandom | grep -o '[:alnum:]' | head -n 30 | tr -d '\n'; echo
```

Here's an even simpler version of the urandom one.

```
“ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c6
```

This one manages to use the very useful `dd` command.

```
“ dd if=/dev/urandom bs=1 count=32 2>/dev/null | base64 -w 0 | rev | cut -b 2-  
| rev
```

You can even create a random left-hand password, which would let you type your password with one hand.

```
“ </dev/urandom tr -dc '12345!@#$%qwertyQWERTasdfgASDFGzxcvZXCVB' | head -c8;  
echo ""
```

If you're going to be using this all the time, it's probably a better idea to put it into a function. In this case, once you run the command once, you'll be able to use *randpw* anytime you want to generate a random password. You'd probably want to put this into your `~/.bashrc` file.

```
“ randpw(){ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c${1:-16};echo;}
```

You can use this same syntax to make any of these into a function—just replace everything inside the `{ }`

And here's the easiest way to make a password from the command line, which works in Linux, Windows with Cygwin, and probably Mac OS X. I'm sure that some people will complain that it's not as random as some of the other options, but honestly, it's random enough if you're going to be using the whole thing.

```
“ date | md5sum
```

Yeah, that's even easy enough to remember.

## bash oneliner commands

\* alle durch salt generierten Icinga config files in einer Datei schreiben und den salt header entfernen

\* apply rules in `/etc/icinga2/zones.d/global-templates/notifications/slack/`:

```
workdir="/etc/icinga2/zones.d/global-templates/notifications/slack";for file in $(ls
${workdir}); do echo ${file}; cat ${workdir}/${file} | sed -E -e '/^#.*|^\/*|^\*\/d';echo
"";echo "" ; done > /tmp/slack.log
```

---

Revision #5

Created 17 July 2018 11:07:11 by David

Updated 14 December 2023 08:04:23 by David