

Если вы видите что-то необычное, просто сообщите мне.

# Готовые скрипты

- Ansible
  - Wait for connection if not available, timeout 600 seconds
- bash
  - Удаление ноды через cron
  - Template
- Установка openvpn(шаблончик превратить в скрипт)
- Пройтись по папкам и выполнить в каждой из них команду

# Ansible

# Wait for connection if not available, timeout 600 seconds

```
- hosts: vivumlab
gather_facts: no
pre_tasks:
  - name: Wait for connection if not available, timeout 600 seconds
    wait_for_connection:
      timeout: 600
      delay: 0
    when: vlab_connection_wait == true
```

```
tasks:
  - name: Restart services
    become: yes
    systemd:
      name: "{{ service_item }}"
      state: restarted
      enabled: true
    when: query | bool
    loop: "{{ services | flatten(1) }}"
    loop_control:
      loop_var: service_item
    ignore_errors: yes
  vars:
    query: "{{ vars[service_item]['enable'] }}"
```

bash

bash

# Удаление ноды через cron

В крон помещаем запись вида:

```
* /5 * * * * /root/deletenode.sh
```

Каждые пять минут запускаем скрипт `/root/deletenode.sh` следующего содержания:

```
#!/bin/bash
LOGGERTAG=nodeDeletion
kubectl --kubeconfig /etc/kubernetes/admin.conf get nodes | grep NotReady | awk '{print $1;}' > last
logger -t $LOGGERTAG "Deleting nodes:"
RESULT=$(kubectl --kubeconfig /etc/kubernetes/admin.conf delete node $(grep -f last old) 2>&1)
logger -t $LOGGERTAG "$RESULT"
mv last old
```

Результат работы скрипта будет записываться в syslog, чтобы посмотреть его, нужно выполнить следующую команду:

```
sudo journalctl -t nodeDeletion
```

bash

# Template

Things Use `bash`. Using `zsh` or `fish` or any other, will make it hard for others to understand / collaborate. Among all shells, bash strikes a good balance between portability and DX.

Just make the first line be `#!/usr/bin/env bash`, even if you don't give executable permission to the script file.

Use the `.sh` (or `.bash`) extension for your file. It may be fancy to not have an extension for your script, but unless your case explicitly depends on it, you're probably just trying to do clever stuff. Clever stuff are hard to understand.

Use `set -o errexit` at the start of your script.

So that when a command fails, bash exits instead of continuing with the rest of the script. Prefer to use `set -o nounset`. You may have a good excuse to not do this, but, my opinion, it's best to always set it.

This will make the script fail, when accessing an unset variable. Saves from horrible unintended consequences, with typos in variable names. When you want to access a variable that may or may not have been set, use `"${VARNAME-}"` instead of `"$VARNAME"`, and you're good. Use `set -o pipefail`. Again, you may have good reasons to not do this, but I'd recommend to always set it.

This will ensure that a pipeline command is treated as failed, even if one command in the pipeline fails. Use `set -o xtrace`, with a check on `$TRACE` env variable.

For copy-paste: `if [ "${TRACE-0}" == "1" ]; then set -o xtrace; fi`. This helps in debugging your scripts, a lot. Like, really lot. People can now enable debug mode, by running your script as `TRACE=1 ./script.sh` instead of `./script.sh`. Use `[[ ]]` for conditions in `if` / `while` statements, instead of `[ ]` or `test`.

`[[ ]]` is a bash builtin keyword, and is more powerful than `[ ]` or `test`. Always quote variable accesses with double-quotes.

One place where it's okay not to is on the left-hand-side of an `[[ ]]` condition. But even there I'd recommend quoting. When you need the unquoted behaviour, using bash arrays will likely serve you much better. Use local variables in functions.

Accept multiple ways that users can ask for help and respond in kind.

Check if the first arg is `-h` or `--help` or `help` or just `h` or even `-help`, and in all these cases, print help text and exit. Please. For the sake of your future-self. When printing error messages, please redirect to `stderr`.

Use `echo 'Something unexpected happened' >&2` for this. Use long options, where possible (like `--silent` instead of `-s`). These serve to document your commands explicitly.

Note though, that commands shipped on some systems like macOS don't always have long options. If appropriate, change to the script's directory close to the start of the script.

And it's usually always appropriate. Use `cd "$(dirname "$0")"`, which works in most cases. Use `shellcheck`. Heed its warnings

## Template

```
#!/usr/bin/env bash

set -o errexit
set -o nounset
set -o pipefail

if [[ "${TRACE-0}" == "1" ]]; then
    set -o xtrace
fi

if [[ "${1-}" =~ ^-*h(elp)?$ ]]; then
    echo 'Usage: ./script.sh arg-one arg-two'
    This is an awesome bash script to make your life better.
    exit
fi

cd "$(dirname "$0")"
main() {
```

```
    echo do awesome stuff  
}  
main "$@"
```

## Conclusion

I try to follow these rules in my scripts, and they're known to have made at least my own life better. I'm still not consistent though, unfortunately, in following my own rules. So perhaps writing them down this way will help me improve there as well.



# Установка openvpn(шаблончик превратить в скрипт)

```
sudo mkdir /etc/openvpn/easy-rsa
sudo rm /etc/openvpn/easy-rsa/ -r
sudo mkdir /etc/openvpn/easy-rsa
sudo ./easyrsa init-pki
sudo ./easyrsa build-ca
sudo ./easyrsa gen-dh
sudo openvpn --genkey --secret /etc/openvpn/easy-rsa/pki/ta.key
sudo ./easyrsa gen-crl
sudo ./easyrsa build-server-full server nopass
sudo cp ./pki/ca.crt /etc/openvpn/ca.crt
sudo cp ./pki/dh.pem /etc/openvpn/dh.pem
sudo cp ./pki/crl.pem /etc/openvpn/crl.pem
sudo cp ./pki/ta.key /etc/openvpn/ta.key
sudo cp ./pki/issued/server.crt /etc/openvpn/server.crt
sudo cp ./pki/private/server.key /etc/openvpn/server.key
sudo vim /etc/openvpn/server.conf
cd ..
sudo openvpn /etc/openvpn/server.conf
sudo systemctl start openvpn@server
sudo sysctl -w net.ipv4.ip_forward=1
#заменить eth0 на нужный интерфейс
sudo iptables -I FORWARD -i tun0 -o eth0 -j ACCEPT
sudo iptables -I FORWARD -i eth0 -o tun0 -j ACCEPT
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
cd easy-rsa/
sudo ./easyrsa build-client-full losst nopass
sudo mkdir -p /etc/openvpn/clients/losst
cd /etc/openvpn/clients/losst
sudo cp /etc/openvpn/easy-rsa/pki/ca.crt /etc/openvpn/clients/losst/
```

```
sudo cp /etc/openvpn/easy-rsa/pki/ta.key /etc/openvpn/clients/losst/
sudo cp /etc/openvpn/easy-rsa/pki/issued/losst.crt /etc/openvpn/clients/losst/
sudo cp /etc/openvpn/easy-rsa/pki/private/losst.key /etc/openvpn/clients/losst/
sudo cp /usr/share/doc/openvpn/examples/sample-config-files/client.conf ./losst.conf
ip -br a
sudo vi ./losst.conf
sudo cat losst.crt
cat /etc/openvpn/clients/losst/losst.conf
cat losst.conf
sudo cat ca.crt
sudo cat losst.key
sudo cat ta.key
sudo cat ca.crt
sudo cat losst.key
sudo cat ta.key
```

# Пройтись по папкам и выполнить в каждой из них команду

```
#!/bin/bash

# Перебираем все папки в текущем каталоге
for dir in */; do

    # Переходим в каждую папку
    cd "$dir"

    # Проверяем наличие артефакта
    marker_file="processed_marker.txt"
    if [ -f "$marker_file" ]; then
        echo "Skipping folder $dir as it's already processed"
        cd ..
        continue
    fi

    # Выводим название папки
    echo "Contents of folder: $dir"

    # Выполняем команду
    tofu init && tofu apply && touch "$marker_file"

    # Возвращаемся обратно в исходную папку
    cd ..
done
```