

□ Шаг 1. смотрим диски

```
lsblk sdb
```

□ Шаг 2. вставить скрипт

[builder.sh](#)

```
1. cat << 'MAIN_EOF' > builder.sh
2. #!/bin/bash
3. set -e
4.
5. #
   =====
   =====
6. # 0. ГЛОБАЛЬНАЯ ТЕХНИЧЕСКАЯ КОНФИГУРАЦИЯ СЕРВЕРА И ССЫЛОК
7. #
   =====
   =====
8. TARGET_FLASH="/dev/sdb"           # Физический диск флешки, размечаемый в
   Ubuntu
9. SERVER_NAME="arch-server"         # Имя хоста (hostname) будущего ПК
10. ADMIN_NAME="eva"                 # Логин администратора с доступом к sudo
11. ADMIN_PASS="65421"               # Пароль администратора
12. ROOT_PASS="rootpassword"         # Пароль суперпользователя (root)
13. OFFLINE_MODE="YES"               # Измените на "NO", если на целевом ПК
   есть интернет
14.
15. # Базовый образ Arch Linux
16. URL_BOOTSTRAP="https://mirror.yandex.ru/archlinux/iso/latest/archl
   inux-bootstrap-x86_64.tar.zst"
17.
18. # Официальные репозитории и зеркала (используются для скачивания и
   прописываются на ПК)
19. URL_YANDEX="https://mirror.yandex.ru/archlinux/\$repo/os/\$arch"
20. URL_PKGS="https://archlinux.pkgs.org/archlinux/\$repo/os/\$arch"
21. URL_RACKSPACE="http://mirror.rackspace.com/archlinux/\$repo/os/\$a
   rch"
22. URL_PKGBUILD="https://geo.mirror.pkgbuild.com/\$repo/os/\$arch"
23.
24. #
   =====
   =====
25. # ШАГ 1: Верификация безопасности накопителей
26. #
   =====
   =====
27. echo "=== ШАГ 1: Верификация безопасности накопителей ==="
28. UBUNTU_ROOT_DISK=$(lsblk -no PKNAME $(findmnt -nvo SOURCE /)
   2>/dev/null || true)
```

```
29. if [ -z "$UBUNTU_ROOT_DISK" ]; then
30.     UBUNTU_ROOT_DISK=$(lsblk -dno NAME,MOUNTPOINTS | grep -E '/'$'
    | awk '{print $1}')
31. fi
32.
33. if [ "/dev/$UBUNTU_ROOT_DISK" == "$TARGET_FLASH" ]; then
34.     echo "КРИТИЧЕСКАЯ ОШИБКА: Запрещено размечать диск запущенной
    Ubuntu ($TARGET_FLASH)!"
35.     exit 1
36. fi
37.
38. echo "Целевой диск определен как безопасный: $TARGET_FLASH"
39. echo "ВНИМАНИЕ! Диск $TARGET_FLASH будет totally очищен через 3 секунд..."
40. sleep 3
41.
42. #
    =====
    =====
43. # ШАГ 2: Принудительное уничтожение метаданных и разметки
44. #
    =====
    =====
45. echo "=== ШАГ 2: Принудительное уничтожение метаданных и разметки ==="
46. # Принудительно отрываем все виртуальные ФС, которые могли остаться в
    памяти хоста
47. sudo umount -l /mnt/arch/dev/pts 2>/dev/null || true
48. sudo umount -l /mnt/arch/dev 2>/dev/null || true
49. sudo umount -l /mnt/arch/proc 2>/dev/null || true
50. sudo umount -l /mnt/arch/sys 2>/dev/null || true
51. sudo umount -l /mnt/arch/boot 2>/dev/null || true
52. sudo umount -l /mnt/arch 2>/dev/null || true
53. sudo umount -l ${TARGET_FLASH}* 2>/dev/null || true
54.
55. # Стираем сигнатуры файловых систем
56. sudo wipefs -a --force "$TARGET_FLASH"
57.
58. sudo dd if=/dev/zero of="$TARGET_FLASH" bs=1M count=20 status=none
59. sudo partprobe "$TARGET_FLASH" || true
60.
61. #
    =====
    =====
62. # ШАГ 3: Создание новой таблицы разделов GPT
63. #
    =====
    =====
64. echo "=== ШАГ 3: Создание новой таблицы разделов GPT ==="
65. printf "g\nn\n1\nn+512M\nt\n1\nnn\n2\n\n\nw\nn" | sudo fdisk --wipe
    always --wipe-partitions always "$TARGET_FLASH"
66. sudo partprobe "$TARGET_FLASH" || true
67.
```

```
68. if [[ "$TARGET_FLASH" == *"nvme"* ]]; then
69.     PART1="${TARGET_FLASH}p1"
70.     PART2="${TARGET_FLASH}p2"
71. else
72.     PART1="${TARGET_FLASH}1"
73.     PART2="${TARGET_FLASH}2"
74. fi
75.
76. echo "Ожидание инициализации разделов ядром..."
77. sudo partprobe /dev/sdb || true
78. sudo udevadm settle || true
79. sleep 2
80.
81. #
=====
=====
82. # ШАГ 4: Безусловное форматирование файловых систем
83. #
=====
=====
84. echo "=== ШАГ 4: Безусловное форматирование файловых систем ==="
85. sudo mkfs.vfat -F 32 "$PART1"
86. sudo mkfs.ext4 -F "$PART2"
87.
88. #
=====
=====
89. # ШАГ 5: Развертывание структуры точек монтирования
90. #
=====
=====
91. echo "=== ШАГ 5: Развертывание структуры точек монтирования ==="
92. sudo mkdir -p /mnt/arch
93. sudo mount "$PART2" /mnt/arch
94. sudo mkdir -p /mnt/arch/boot
95. sudo mount "$PART1" /mnt/arch/boot
96.
97. #
=====
=====
98. # ШАГ 6: Скачивание официального образа Arch Linux
99. #
=====
=====
100. echo "=== ШАГ 6: Скачивание официального образа Arch Linux ==="
101. sudo apt update && sudo apt install -y zstd wget
102. if [ ! -f archlinux-bootstrap-x86_64.tar.zst ]; then
103.     wget --timeout=15 --tries=3 "$URL_BOOTSTRAP"
104. fi
105. sudo tar -I zstd -xf archlinux-bootstrap-x86_64.tar.zst --strip-components=1 -C /mnt/arch
```

```
106.
107. #
=====
=====
108. # ШАГ 7: Статическая генерация таблицы fstab флешки
109. #
=====
=====
110. echo "=== ШАГ 7: Статическая генерация таблицы fstab флешки ==="
111. EFI_UUID=$(sudo blkid -s UUID -o value "$PART1")
112. ROOT_UUID=$(sudo blkid -s UUID -o value "$PART2")
113. sudo mkdir -p /mnt/arch/etc
114. printf "UUID=%s /boot vfat defaults 0 2\nUUID=%s / ext4 defaults 0
1\n" "$EFI_UUID" "$ROOT_UUID" | sudo tee /mnt/arch/etc/fstab
115.
116. #
=====
=====
117. # ШАГ 8: Внедрение скрипта АВТОУСТАНОВКИ на флешку (ОФФЛАЙН МЕТОД)
118. #
=====
=====
119. echo "=== ШАГ 8: Внедрение скрипта АВТОУСТАНОВКИ на флешку ==="
120. sudo mkdir -p /mnt/arch/root
121.
122. sudo cat << EOF | sudo tee /mnt/arch/root/forest-install.sh >
/dev/null
123. #!/bin/bash
124. set -e
125. echo "=== КОМБАЙН ЗАПУЩЁН ==="
126.
127. MY_DISK=$(lsblk -no PKNAME $(findmnt -nvo SOURCE /) 2>/dev/null
|| true)
128. if [ -z "\$MY_DISK" ]; then
129.     MY_DISK=$(lsblk -no PKNAME /bootmnt 2>/dev/null || echo
"sdb")
130. fi
131.
132. TARGET_DISK=$(lsblk -dno NAME,TYPE | grep -v "loop" | grep -v
"rom" | grep -v "\$MY_DISK" | head -n1 | awk '{print \$1}')
133. TARGET="/dev/\$TARGET_DISK"
134. if [ -z "\$TARGET_DISK" ]; then
135.     echo "КРИТИЧЕСКАЯ ОШИБКА: Жёсткий диск ПК не найден!"
136.     exit 1
137. fi
138.
139. echo "Целевой жёсткий диск ПК определён: \$TARGET. Уничтожаем старые
данные..."
140. umount -f \$TARGET* 2>/dev/null || true
141. wipefs -a --force "\$TARGET"
142. dd if=/dev/zero of="\$TARGET" bs=1M count=20 status=none
```

```
143. partprobe "\$TARGET" || true
144.
145. echo "Разметка жёсткого диска ПК..."
146. printf "g\n\n1\n\n+512M\nt\n1\n\n2\n\n\nw\n" | fdisk --wipe
    always --wipe-partitions always "\$TARGET"
147. if [[ "\$TARGET" == *"nvme"* ]]; then
148.     TPART1="\${TARGET}p1"
149.     TPART2="\${TARGET}p2"
150. else
151.     TPART1="\${TARGET}1"
152.     TPART2="\${TARGET}2"
153. fi
154.
155. echo "Форматирование жёсткого диска ПК..."
156. mkfs.vfat -F 32 "\$TPART1"
157. mkfs.ext4 -F "\$TPART2"
158.
159. echo "Монтирование дисков ПК под установку..."
160. mkdir -p /mnt/target && mount "\$TPART2" /mnt/target
161. mkdir -p /mnt/target/boot && mount "\$TPART1" /mnt/target/boot
162.
163. #
    =====
    =====
164. # ШАГ 8.1 ОФЛАЙН УСТАНОВКА С ОБНОВЛЕНИЕМ
165. #
    =====
    =====
166. #
    =====
    =====
167. # НАСТОЯЩИЙ АВТОНОМНЫЙ ОФЛАЙН-МЕТОД 20.05.26 14:15
168. #
    =====
    =====
169. echo "Создаем необходимые папки под локальный кэш на жестком диске ПК..."
170. mkdir -p /mnt/target/var/cache/pacman/pkg
171. mkdir -p /mnt/target/var/lib/pacman/sync
172. mkdir -p /mnt/target/etc/pacman.d
173.
174. echo "Копируем автономный кэш пакетов и базы прямо с флешки на диск ПК..."
175. if [ -d "/var/cache/pacman/pkg" ]; then
176.     cp -R /var/cache/pacman/pkg/.
        /mnt/target/var/cache/pacman/pkg/
177. fi
178. if [ -d "/var/lib/pacman/sync" ]; then
179.     cp -R /var/lib/pacman/sync/. /mnt/target/var/lib/pacman/sync/
180. fi
181.
182.
183. cp /etc/pacman.conf /mnt/target/etc/pacman.conf
```

```
184. touch /mnt/target/etc/pacman.d/mirrorlist
185.
186. echo "Развертывание системы из локальных файлов без интернета..."
187. # Подставляем имена файлов через find прямо на ПК, защищая строку от
    Ubuntu
188. pacman -U --noconfirm --root /mnt/target --dbpath
    /mnt/target/var/lib/pacman \$(find
    /mnt/target/var/cache/pacman/pkg/ -name "*.pkg.tar.zst")
189.
190. #
    =====
    =====
191. # КОНЕЦ НАСТОЯЩИЙ АВТОНОМНЫЙ ОФЛАЙН-МЕТОД 20.05.26 14:15
192. #
    =====
    =====
193.
194. echo "Генерация таблицы fstab целевого ПК..."
195. printf "UUID=\$(blkid -s UUID -o value \${TPART2}) / ext4 defaults 0
    1\nUUID=\$(blkid -s UUID -o value \${TPART1}) /boot vfat defaults 0
    2\n" > /mnt/target/etc/fstab
196.
197. echo "Вход в chroot окружение ПК..."
198. mount --types proc /proc /mnt/target/proc
199. mount --rbind /sys /mnt/target/sys
200. mount --make-rslave /mnt/target/sys
201. mount --rbind /dev /mnt/target/dev
202. mount --make-rslave /mnt/target/dev
203. mount --rbind /dev/pts /mnt/target/dev/pts
204.
205.
206.
207. arch-chroot /mnt/target /bin/bash << 'CHROOT_INNER_EOF'
208. set -e
209. # Отключите автоматическую синхронизацию
210. # Автоматическая настройка времени без пользовательского ввода
211. # Вариант 1: Использовать UTC (рекомендуется)
212. #echo "UTC" > /etc/timezone
213. #ln -sf /usr/share/zoneinfo/UTC /etc/localtime
214. #timedatectl set-local-rtc 0
215.
216. # Вариант 2: Использовать локальное время с материнки напрямую через конфиги
217. hwclock --systohc --localtime
218. mkdir -p /etc/systemd
219. mkdir -p /run/systemd/system
220. mkdir -p /run/systemd/catalog/
221.
222. touch /etc/machine-id
223.
224. echo -e "[Time]\nLocalRTC=yes" > /etc/adjtime
225. echo "Конфигурация пользователей..."
```

```
226. useradd -m -G wheel -s /bin/bash "${ADMIN_NAME}"
227.
228. #
=====
=====
229. # ЖЕСТКОЕ ПОДАВЛЕНИЕ СИНЕГО ЭКРАНА НА ЦЕЛЕВОМ ПК
230. #
=====
=====
231. echo "en_US.UTF-8 UTF-8" > /etc/locale.gen
232. locale-gen
233. echo "LANG=en_US.UTF-8" > /etc/locale.conf
234. echo "KEYMAP=us" > /etc/vconsole.conf
235. echo "${SERVER_NAME}" > /etc/hostname
236. systemd-machine-id-setup
237. ln -sf /dev/null /etc/systemd/system/systemd-firstboot.service
238.
239. #
=====
=====
240.
241. echo "Конфигурация пользователей..."
242. echo "${ADMIN_NAME}:${ADMIN_PASS}" | chpasswd
243. echo "root:${ROOT_PASS}" | chpasswd
244. echo "%wheel ALL=(ALL:ALL) ALL" > /etc/sudoers.d/wheel
245.
246. touch /etc/locale.conf
247. touch /etc/vconsole.conf
248. touch /etc/machine-id
249.
250. echo "Активация служб..."
251. sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin no/'
    /etc/ssh/sshd_config
252. systemctl enable sshd NetworkManager
253.
254. echo "Установка загрузчика GRUB..."
255. grub-install --target=x86_64-efi --efi-directory=/boot --
    bootloader-id=B00T --removable --recheck
256. grub-mkconfig -o /boot/grub/grub.cfg
257. CHROOT_INNER_EOF
258.
259. printf "## Russia\nServer = %s\nServer = %s\n##
    International\nServer = %s\nServer = %s\n" \
260. "${URL_YANDEX}" "${URL_PKGS}" "${URL_RACKSPACE}" "${URL_PKGBUILD}"
    > /mnt/target/etc/pacman.d/mirrorlist
261.
262. # Размонтируем за собой виртуальные ФС на флешке
263. umount -l /mnt/target/dev/pts
264. umount -l /mnt/target/dev
265. umount -l /mnt/target/sys
266. umount -l /mnt/target/proc
```

```
267. umount -R /mnt/target
268.
269. umount -R /mnt/target
270. echo "Система успешно развёрнута автономно! Перезагрузка через 3 секунды..."
271. sleep 3
272. reboot
273. EOF
274.
275. # Настройка прав доступа к файлу автоустановщика
276. sudo chmod 755 /mnt/arch/root
277. sudo chmod +x /mnt/arch/root/forest-install.sh
278. sudo chmod 700 /mnt/arch/root
279.
280. #
=====
=====
281. # ШАГ 9: Создание и регистрация фоновой службы systemd
282. #
=====
=====
283. echo "=== ШАГ 9: Создание и регистрация фоновой службы systemd ==="
284. sudo cat << 'SERVICE_EOF' | sudo tee
    /mnt/arch/etc/systemd/system/forest-autoinstall.service >
    /dev/null
285. [Unit]
286. Description=Автоустановщик Arch Linux без монитора
287. After=multi-user.target
288. [Service]
289. Type=idle
290. ExecStart=/root/forest-install.sh
291. StandardOutput=tty
292. StandardError=tty
293. [Install]
294. WantedBy=multi-user.target
295. SERVICE_EOF
296.
297. sudo mkdir -p /mnt/arch/etc/systemd/system/multi-user.target.wants
298. sudo ln -sf /etc/systemd/system/forest-autoinstall.service
    /mnt/arch/etc/systemd/system/multi-user.target.wants/forest-
    autoinstall.service
299.
300. #
=====
=====
301. # ШАГ 10: Формирование чистых репозиториях для флешки
302. #
=====
=====
303. echo "=== ШАГ 10: Формирование чистых репозиториях для флешки ==="
304. sudo mkdir -p /mnt/arch/etc/pacman.d
305.
```



```
306. # Пишем зеркала через стандартный echo, полностью избегая капризных
    Here-Doc с пайпами
307. echo "Server = ${URL_YANDEX}" | sudo tee
    /mnt/arch/etc/pacman.d/mirrorlist > /dev/null
308. echo "Server = ${URL_PKGS}" | sudo tee -a
    /mnt/arch/etc/pacman.d/mirrorlist > /dev/null
309.
310. #
    =====
    =====
311. # ШАГ 11: Подготовка локального репозитория на флешке (Выполняет Ubuntu)
312. #
    =====
    =====
313. echo "=== ШАГ 11: Подготовка локального репозитория силами Ubuntu ==="
314. sudo mkdir -p /mnt/arch/var/cache/pacman/pkg
315. sudo mkdir -p /mnt/arch/var/lib/pacman
316.
317. # Генерируем правильный pacman.conf для флешки (пока со стандартными
    зеркалами)
318. cat << 'EOF' | sudo tee /mnt/arch/etc/pacman.conf > /dev/null
319. [options]
320. Architecture = auto
321. SigLevel = Optional TrustAll
322. LocalFileSigLevel = Optional
323.
324. [core]
325. Include = /etc/pacman.d/mirrorlist
326.
327. [extra]
328. Include = /etc/pacman.d/mirrorlist
329. EOF
330.
331. # Скачиваем базы данных и BCE пакеты, зайдя в chroot флешки (тут есть
    библиотеки и интернет)
332. sudo /mnt/arch/usr/bin/arch-chroot /mnt/arch /bin/bash <<
    'CHROOT_EOF'
333.
334. set -e
335.
336. echo "Инициализация ключей внутри окружения флешки..."
337. pacman-key --init
338. pacman-key --populate archlinux
339.
340. echo "Скачиваем базы данных и BCE пакеты в автономный кэш флешки..."
341. pacman -Syw --noconfirm \
342.     iana-etc filesystem linux-api-headers tzdata licenses \
343.     glibc libgcc libstdc++ libasan libatomic libgfortran libgomp \
344.     liblsan libobjc libquadmath libtsan libubsan gcc-libs \
345.     ncurses readline bash acl attr gmp zlib sqlite \
346.     util-linux-libs e2fsprogs keyutils gdbm brotli xz \
```

```
347. lz4 zstd openssl libsasl libldap libevent libverto lmbd \
348. krb5 libcap-ng audit libxcrypt libtirpc libnsl pambase \
349. libpgp-error libgcrypt systemd-libs pam libcap coreutils \
350. bzip2 libseccomp file findutils mpfr gawk pcre2 grep \
351. procs-ng sed tar libtasn1 libffi libp11-kit p11-kit \
352. ca-certificates-utils ca-certificates-mozilla ca-certificates \
353. libunistring libidn2 libnghttp2 libnghttp3 nettle leancrypto \
354. gnutls libngtcp2 libpsl libssh2 curl json-c gnu lib-l10n icu \
355. libxml2 gettext hwd data kmod pciutils psmisc shadow util-linux \
356. gzip licenses libksba libusb libassuan libsysprof-capture \
357. glib2 tpm2-tss libsecret pinentry npth gnupg gpgme libarchive \
358. pacman-mirrorlist device-mapper popt cryptsetup expat \
359. dbus dbus-broker dbus-broker-units dbus-units kbd libelf \
360. systemd jansson binutils libmakepkg-dropins pacman \
361. archlinux-keyring systemd-sysvcompat iputils libmnl \
362. libnftlink libnetfilter_conntrack libnftnl libnl libpcap \
363. nftables iptables libbpf iproute2 base mkinitcpio-busybox \
364. diffutils mkinitcpio \
365. linux linux-firmware-whence linux-firmware-amdgpu \
366. linux-firmware-atheros linux-firmware-broadcom \
367. linux-firmware-cirrus linux-firmware-intel \
368. linux-firmware-mediatek linux-firmware-nvidia \
369. linux-firmware-other linux-firmware-radeon \
370. linux-firmware-realtek linux-firmware libedit \
371. openssh libmm-glib libndp gpm pcre slang libnewt nspr nss \
372. libnm libdaemon libsodium libpgm zeromq libteam \
373. mobile-broadband-provider-info duktape polkit \
374. pcsclite wpa_supplicant networkmanager sudo \
375. nano dosfstools mtools mailcap nginx libmd libbsd \
376. tallocc tevent tdb ldb avahi libcups liburing libwbclient \
377. mpdecimal python cifs-utils smbclient samba grub efivar \
378. efibootmgr libzip argon2 oniguruma php php-fpm
379.
380.
381.
382.
383.
384. CHROOT_EOF
385.
386. # Переводим pacman.conf флешки на полностью локальный оффлайн-режим
    file:///
387. cat << 'EOF' | sudo tee /mnt/arch/etc/pacman.conf > /dev/null
388. [options]
389. Architecture = auto
390. SigLevel = Optional TrustAll
391. LocalFileSigLevel = Optional
392.
393. [core]
394. Server = file:///var/cache/pacman/pkg
395.
396. [extra]
```

```
397. Server = file:///var/cache/pacman/pkg
398. EOF
399.
400.
401. #
=====
=====
402. # ШАГ 12: Локальная автономная установка на флешку (БЕЗ ИНТЕРНЕТА)
403. #
=====
=====
404. echo "=== ШАГ 12: Накатывание базовой системы на флешку из локального кэша
=== "
405.
406. # Генерируем базовые конфиги флешки до chroot, чтобы заблокировать синий
экран
407. echo "en_US.UTF-8 UTF-8" | sudo tee /mnt/arch/etc/locale.gen >
/dev/null
408. echo "LANG=en_US.UTF-8" | sudo tee /mnt/arch/etc/locale.conf >
/dev/null
409. echo "KEYMAP=us" | sudo tee /mnt/arch/etc/vconsole.conf >
/dev/null
410. echo "arch-flash" | sudo tee /mnt/arch/etc/hostname > /dev/null
411.
412. # Принудительная инициализация идентификаторов системы
413. sudo systemd-machine-id-setup --root=/mnt/arch
414. sudo touch /mnt/arch/etc/machine-id
415.
416. # ЖЕСТКОЕ ОФФЛАЙН ПОДАВЛЕНИЕ FIRSTBOOT (Записываем строго внутрь
флешки с sudo)
417. sudo mkdir -p /mnt/arch/etc/systemd/system
418. sudo ln -sf /dev/null /mnt/arch/etc/systemd/system/systemd-
firstboot.service
419. sudo touch /mnt/arch/var/lib/systemd/clock
420.
421. # РЕШЕНИЕ ПРОБЛЕМЫ 3/13 И 12/13: Создаем каталоги и монтируем виртуальные
ФС ядра хоста
422. sudo mkdir -p /mnt/arch/usr/lib/systemd/catalog
423.
424. echo "Развертываем систему на флешке из автономного кэша..."
425. # Используем встроенный chroot флешки, передавая команды через bash -c,
чтобы не ломать /dev/stdin
426. sudo /mnt/arch/usr/bin/arch-chroot /mnt/arch /bin/bash -c "
427. set -e
428. # Ставим напрямую по маске файлы пакетов
429. pacman -U --noconfirm /var/cache/pacman/pkg/*.pkg.tar.zst
430.
431. # Настраиваем GRUB на флешке как съемный диск
432. grub-install --target=x86_64-efi --efi-directory=/boot --
bootloader-id=Arch_Flash --removable --recheck
433. grub-mkconfig -o /boot/grub/grub.cfg
```

```
434. "  
435.  
436. #  
=====
```

```
437. # ШАГ 13: финальная проверка  
438. #  
=====
```

```
439. echo "=== ШАГ 13: финальная проверка ==="  
440.  
441. sudo umount -l /mnt/arch/boot 2>/dev/null || true  
442. sudo umount -l /mnt/arch 2>/dev/null || true  
443.  
444.  
445.  
446. MAIN_EOF
```

□ Шаг 4. Очистите его от символов Windows

```
sed -i 's/\r$//' builder.sh
```

□ Шаг 5. Разрешите запуск

```
chmod +x builder.sh && ./builder.sh
```

□ Шаг 4. запустить скрипт

```
./builder.sh
```

From:

<https://wwoss.direct.quickconnect.to/> - **worldwide open-source software**

Permanent link:

<https://wwoss.direct.quickconnect.to/doku.php?id=tmp>

Last update: **2026/05/22 12:15**

