

Linux: terminal, powłoka i skrypty

Technologie Informacyjne i Komunikacyjne R

Michał Parniak

`mparniak@fuw.edu.pl` `www.qodl.eu`

Na podstawie materiałów: Katarzyna Grzelak („Linux – podstawy”, „Linux – skrypty”)

semestr zimowy 2026/27

Zasady zaliczenia

Cztery bloki — zaliczenie w czasie ćwiczeń,
każdy **10%** oceny:

Linux (terminal, skrypty)	14.10
L ^A T _E X	23.10
Mathematica	6.11
Python (analiza danych)	27.11

Pomiar sterowany przez komputer — **60%**
oceny:

1. zlutowanie prostego układu pomiarowego,
2. program sterujący pomiarem (Python),
3. pomiary,
4. opracowanie danych z analizą niepewności (Python),
5. sprawozdanie w L^AT_EX-u (wykresy: Matplotlib)
+ rozmowa o sprawozdaniu.

Warunek zaliczenia

Sprawozdanie musi być zaliczone na co
najmniej **50%**.

Skala ocen

> 98%	bdb+	> 75%	db
> 90%	bdb	> 62,5%	dst+
> 82,5%	db+	> 50%	dst

- Do sprawozdania potrzebny jest materiał z *Analizy Niepewności Pomiarowych*.
- W piątek 13.11 obowiązuje środowy plan zajęć.

Szczegóły:

users.fuw.edu.pl/~kaste/tikr.html

Plan wykładu

Terminal i powłoka

Pliki i katalogi

Strumienie, potoki, wyszukiwanie

Zmienne i konfiguracja powłoki

Skrypty

Terminal, powłoka, polecenie

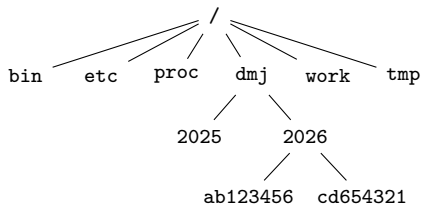
- **Terminal** — okno, w którym piszemy tekst.
- **Powłoka** (*shell*, u nas `bash`) — program, który czyta polecenia, uruchamia je i pokazuje wynik.
- Te same polecenia można zapisać w pliku: **skrypt** — prosty język programowania.
- Po co? Automatyzacja, praca zdalna (`ssh`), klastry obliczeniowe, aparatura w laboratorium.

Składnia: polecenie -opcje
argumenty

```
ls
ls -l
ls -l -a /etc      # = ls -la /etc
ls --help
man ls             # q = wyjście
```

- Wielkie i małe litery są **rozróżniane**.
- `#` rozpoczyna komentarz.

Drzewo katalogów



- Jedno drzewo, korzeń /; nie ma „dysków C:, D:”.
- Ścieżka **bezwzględna**: od /, np. /dmj/2026/ab123456.
- Ścieżka **względna**: od katalogu bieżącego, np. tikr/notatki.txt.

.	katalog bieżący
..	katalog nadrzędny
~	mój katalog domowy (\$HOME)
~kaste	katalog domowy użytkownika kaste

Nazwy plików

- Kropka nie jest wyróżniona; rozszerzenia to tylko **konwencja**: `.txt`, `.tex`, `.pdf`, `.py`, `.cpp`, `.sh`...
- Pliki **ukryte** zaczynają się od kropki (`.bashrc`, `.ssh/`) — zwykle konfiguracja; widać je dopiero w `ls -a`.
- W nazwach unikamy **spacji i polskich liter** (działa, ale utrudnia życie w terminalu i w skryptach).
- Zwyczajowe katalogi: `bin` — programy, `etc` — konfiguracja, `home` — katalogi domowe, `proc` — informacje o systemie i procesach, `tmp` — pliki tymczasowe.

Konto w OKWF

- Katalog **domowy**: /dmj/RRRR/ab123456 — robione są kopie zapasowe.
- Katalog **roboczy**: /work/RRRR/ab123456 — więcej miejsca, bez kopii; w katalogu domowym jest do niego skrót _work_.
- Kopie migawkowe (ostatnia godzina, dzień, tydzień): /dmj/.snapshot/.
- quota — ile miejsca mam i ile zajmuję.
- Z zewnątrz wchodzimy tylko przez tempac.

```
ln -s /work/2026/ab123456 _work_  
quota  
ssh ab123456@tempac.okwf.fuw.edu.pl  
ssh pracownia  
scp plik.txt \  
    ab123456@tempac.okwf.fuw.edu.pl:~  
logout
```

Szczegóły: instrukcja „Dostęp do komputerów na Wydziale Fizyki”.

Adres IP: ip address

```
$ ip a # = ip address
1: lo: <LOOPBACK,UP> ...
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,UP> ...
    link/ether 3c:52:82:1a:2b:3c
    inet 192.168.1.23/24 scope global eth0
    inet6 fe80::3e52:82ff:fe1a:2b3c/64

hostname -I # same adresy
ping -c 3 fuw.edu.pl # czy odpowiada?
curl ifconfig.me # mój adres publiczny
```

(wynik skrócony) lo — interfejs pętli zwrotnej,
eth0/wlp2s0 — karta przewodowa/Wi-Fi; link/ether
— adres MAC karty; inet — IPv4, inet6 — IPv6.

IPv4: 32 bity = 4 bajty, każdy 0–255:

192.168.1.23 = 11000000.10101000.00000001.00010111
/24 — pierwsze 24 bity to adres **sieci** (192.168.1), reszta
to numer **komputera** w sieci (254 możliwe; .255 =
rozgłoszenie).

10.0.0.0/8	prywatne
172.16.0.0/12	prywatne
192.168.0.0/16	prywatne (domowe routery)
127.0.0.0/8	ten komputer (localhost)
169.254.0.0/16	brak serwera DHCP
pozostałe	publiczne

- **Prywatne** adresy powtarzają się w milionach sieci i nie są widoczne z internetu; router zamienia je na swój **publiczny** adres (NAT).
- **Publiczne** są unikalne na świecie, przydziela je dostawca internetu.
- **IPv6:** 128 bitów, zapis szesnastkowy, np.
2001:db8::1; fe80:: — tylko w sieci lokalnej.

Ułatwienia przy wpisywaniu

TAB	uzupełnij nazwę pliku/polecenia
TAB TAB	pokaż możliwości
↑, ↓	poprzednie polecenia
Ctrl+R	szukaj w historii
!!	powtórz ostatnie polecenie
history	lista wydanych poleceń
Ctrl+C	przerwij działający program
Ctrl+D	koniec danych / wyloguj
Ctrl+L	wyczyść ekran

Zasada

Nie przepisujemy długich nazw ręcznie — **TAB** jest szybszy i nie robi literówek.

Uwaga

`rm` nie przenosi do kosza: plik znika od razu.
W OKWF ratunkiem są kopie migawkowe.

Podstawowe polecenia

<code>pwd</code>	gdzie jestem?	<code>cp skąd dokąd</code>	kopiuj
<code>ls, ls -la</code>	zawartość katalogu	<code>cp -r kat dokąd</code>	kopiuj katalog
<code>ls -lt, ls -lh</code>	wg czasu / rozmiary w kB, MB	<code>mv stara nowa</code>	zmień nazwę
<code>cd kat</code>	przejdź do kat	<code>mv plik kat/</code>	przenieś
<code>cd .., cd</code>	w górę / do domu	<code>rm plik</code>	usuń plik
<code>mkdir kat</code>	utwórz katalog	<code>rm -r kat</code>	usuń katalog z zawartością
<code>mkdir -p a/b/c</code>	...z pośrednimi	<code>rmdir kat</code>	usuń pusty katalog
<code>touch plik</code>	utwórz pusty plik	<code>ln -s cel skrót</code>	dowiązanie symboliczne

- Opcja `-i` (`cp -i`, `mv -i`, `rm -i`) pyta przed nadpisaniem/usunięciem.
- Programy są w katalogach z listy `$PATH` — dlatego `ls` zamiast `/bin/ls`. Gdzie leży program?
`which ls`.

Zawartość plików i zajętość dysku

<code>cat plik</code>	wypisz cały plik
<code>less plik</code>	przeglądaj (q wyjście)
<code>head -n 5 plik</code>	pierwsze 5 linii
<code>tail -n 5 plik</code>	ostatnie 5 linii
<code>wc -l plik</code>	liczba linii (-w słów)
<code>cat a b > ab</code>	sklej pliki

<code>du -sh kat</code>	ile zajmuje katalog
<code>du -s *</code>	... każdy element osobno
<code>df -h</code>	wolne miejsce na dyskach

Edytory tekstu

- w terminalu: `nano plik` (Ctrl+O zapisz, Ctrl+X wyjdź), `vim`;
- okienkowe: `code`, `gnome-text-editor`.

Plik tekstowy = ciąg znaków podzielony na linie.

- Ostatnia linia też powinna kończyć się znakiem nowej linii (ENTER na końcu).
- Word/LibreOffice **nie** służą do edycji skryptów.

Wzorce nazw plików (*globbing*)

*	dowolny ciąg znaków (też pusty)
?	dokładnie jeden znak
[an]	jeden ze znaków a, n
[a-n]	jeden znak z zakresu a-n

```
ls -l *.conf
ls -l *.??
ls -l [a-n]*
cp /proc/*info* ~/tikr/
```

Kto rozwija gwiazdkę?

Powłoka, zanim uruchomi polecenie.
ls *.txt dostaje gotową listę nazw.
Sprawdź: echo *

- * nie pasuje do plików ukrytych (z kropką na początku).
- Gdy nic nie pasuje, wzorzec zostaje przekazany dosłownie.

Prawa dostępu

```
-rwxr-x--- 1 ab123456 students 2048 Oct 6 10:12 skrypt.sh
drwx----- 3 ab123456 students 4096 Oct 6 10:15 tkr
```

- 1. znak: - plik, d katalog, l dowiązanie.
- Trzy trójki rwx: **u**żytkownik (właściciel), **g**rupa, **o**—pozostali (*others*); **a** = wszyscy (*all*).
- Dalej: właściciel (ab123456) i grupa (students).
- Prawa zmienia właściciel (lub administrator).

	plik	katalog
r	czytanie	listowanie nazw
w	zmiana	tworzenie/usuwanie plików
x	wykonanie	wejście, dostęp do plików

- Usunięcie pliku zależy od praw do **katalogu**, nie do pliku.
- Bez x na katalogu nie dostaniemy się do żadnego pliku w nim, nawet czytelnego.

chmod — zapis symboliczny

chmod *kto operacja prawa pliki*

kto: u g o a

+ dodaj, - odbierz, = ustaw

	przed		po	
chmod u+x skrypt.sh	-rw-r--r--	->	-rwxr--r--	# mogę uruchomić
chmod +x skrypt.sh	-rw-r--r--	->	-rwxr-xr-x	# wszyscy mogą uruchomić
chmod go-rwx notatki.txt	-rw-r--r--	->	-rw-----	# tylko dla mnie
chmod a-w wyniki.dat	-rw-r--r--	->	-r--r--r--	# ochrona przed nadpisaniem
chmod g+w wspolny.txt	-rw-r--r--	->	-rw-rw-r--	# grupa może edytować
chmod u=rw,go=r raport.pdf	-rwx-----	->	-rw-r--r--	# ustaw wszystko naraz
chmod o-x ~	drwxr-xr-x	->	drwxr-xr--	# nikt obcy nie wejdzie

Rekurencyjnie — cały katalog z zawartością:

```
chmod -R go-rwx ~/tikr
chmod -R u+rwx,go= sum
```

- Wielkie X: x tylko dla katalogów (i plików już wykonywalnych) — nie robimy z każdego pliku programu.
- Sprawdzamy zawsze: `ls -l`,
`ls -ld kat`.

chmod — zapis liczbowy i typowe ustawienia

$r = 4$, $w = 2$, $x = 1$;
sumujemy w każdej trójce:

cyfra	prawa	
7	rwX	4+2+1
6	rw-	4+2
5	r-X	4+1
4	r--	4
0	---	

chmod 750 kat: $u=7$, $g=5$,
 $o=0$
 $\Rightarrow$ `rwXr-x---`

polecenie	wynik	do czego
chmod 755 skrypt.sh	rwXr-xr-x	skrypt dla wszystkich
chmod 700 skrypt.sh	rwX-----	skrypt tylko dla mnie
chmod 644 dane.txt	rw-r--r--	zwykły plik
chmod 600 dane.txt	rw-----	plik prywatny
chmod 444 dane.txt	r--r--r--	tylko do odczytu
chmod 755 kat	rwXr-xr-x	katalog dla innych
chmod 700 kat	rwX-----	katalog prywatny
chmod 600 ~/.ssh/id_rsa	rw-----	klucz prywatny SSH

- Liczbowo ustawiamy **wszystkie** prawa naraz; symbolicznie zmieniamy tylko wybrane.
- ssh odmówi użycia klucza czytelnego dla innych.

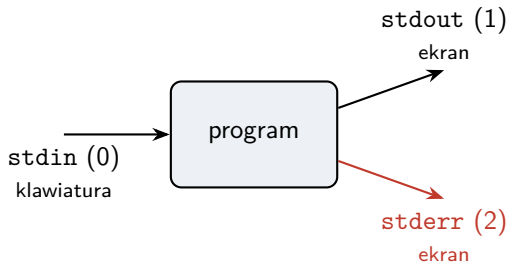
Archiwa i kompresja

<code>tar cvf a.tar kat</code>	spakuj (c reate)
<code>tar czvf a.tar.gz kat</code>	spakuj i skompresuj (g z ip)
<code>tar tvf a.tar</code>	pokaż zawartość (l ist)
<code>tar xvf a.tar</code>	rozpakuj (e xtract)
<code>tar xzvf a.tar.gz</code>	rozpakuj <code>.tar.gz</code>

<code>gzip plik, gunzip plik.gz</code>	kompresja jednego pliku
<code>zip -r a.zip kat</code>	archiwum zip
<code>unzip a.zip</code>	rozpakuj zip

- `v` = *verbose* (wypisuj nazwy), `f` = nazwa archiwum podana dalej — **musi być ostatnia** przed nazwą.
- `.tar` skleja pliki w jeden; dopiero `gzip` zmniejsza rozmiar.
- `.tgz` = `.tar.gz`.
- Rozpakowanie do innego katalogu:
`tar xzf a.tar.gz -C kat.`

Trzy strumienie każdego programu



- **stdin**: dane wejściowe programu.
- **stdout**: wyniki.
- **stderr**: komunikaty o błędach (diagnostyczne).
- Domyślnie oba wyjścia trafiają do terminala — ale każde można **przekierować** osobno.

Przekierowania

<code>pol > plik</code>	stdout do pliku (nadpisz)
<code>pol >> plik</code>	stdout na koniec pliku
<code>pol < plik</code>	stdin z pliku
<code>pol 2> plik</code>	stderr do pliku
<code>pol > p1 2> p2</code>	każde osobno
<code>pol > plik 2>&1</code>	oba do jednego pliku
<code>pol &> plik</code>	to samo, krócej
<code>pol 2> /dev/null</code>	wyrzucić błędy

```
ls -l > lista.txt  
ls -l ~/_work_ >> lista.txt  
wc -l < lista.txt  
find / -name README 2> /dev/null
```

`/dev/null`

„Czarna dziura”: wszystko, co tam zapiszemy, znika.

Uwaga

`sort plik > plik` **kasuje** plik, zanim `sort` go przeczyta.

Potok | łączy stdout jednego programu ze stdin następnego — bez plików pośrednich.

```
ls -l | wc -l
ls | head -5
ps -ef | grep $USER
cat plik.txt | wc -l    # = wc -l < plik.txt
ls -l /bin | tee bin.txt | wc -l
```

- Każdy program robi jedną rzecz dobrze, a potoki je składają — filozofia Uniksa.
- tee plik zapisuje strumień do pliku i przekazuje go dalej.
- Długie potoki budujemy krok po kroku, sprawdzając wynik każdego etapu.

Filtry: sortowanie, wycinanie, liczenie

<code>sort</code>	sortuj alfabetycznie
<code>sort -n, -r</code>	liczbowo / odwrotnie
<code>sort -t: -k3</code>	wg 3. pola (sep. :)
<code>uniq</code>	usuń powtórzone linie
<code>uniq -c</code>	... i policz je
<code>cut -d: -f1,5</code>	pola 1 i 5 (sep. :)
<code>cut -f2</code>	pole 2 (sep. tabulator)
<code>wc -l</code>	policz linie
<code>head, tail</code>	początek / koniec

Co zajmuje najwięcej miejsca?

```
| du -s * | sort -n | tail -1
```

Loginy i imiona z bazy użytkowników:

```
| cut -d: -f1,5 /etc/passwd | sort
```

Najczęstsze słowa w pliku:

```
| tr ' ' '\n' < tekst.txt | sort \  
| uniq -c | sort -n | tail
```

\ na końcu linii: polecenie ciągnie się dalej.

find — szukanie plików po nazwie i atrybutach

```
find / -name README
find . -type f          # zwykłe pliki
find . -type d          # katalogi
find /etc -name '*conf*'
find . -name '*.txt' -mtime -7
find . -size +10M
find . -name '*.tmp' -delete
```

- Składnia: `find gdzie warunki;` przeszukuje wszystkie podkatalogi.
- Wzorzec w **apostrofach** — inaczej rozwinie go powłoka, zanim `find` go zobaczy.
- `-mtime -7`: modyfikowane w ciągu 7 dni.
- Brak praw do katalogów daje dużo błędów: `2> /dev/null`.
- Więcej: `man find`.

grep — szukanie tekstu wewnątrz plików

```
grep abc *           # pliki w kat. bieżącym
grep -i abc *        # Abc = abc = ABC
grep -r abc kat      # rekurencyjnie
grep -rl abc kat     # tylko nazwy plików
grep -n begin plik   # z numerami linii
grep -v '^#' plik    # linie BEZ wzorca
grep -c abc plik     # ile linii pasuje
```

- find szuka po **nazwie**, grep po **treści**.
- grep jest też filtrem w potoku:

```
ps -ef | grep root
grep -in begin p.txt | cut -d: -f1
```

- Wzorzec to wyrażenie regularne: ^
początek linii, \$ koniec, . dowolny znak.

Procesy

UID	PID	PPID	C	STIME	TTY	TIME	CMD
ab123456	4711	4700	0	10:02	pts/0	00:00:00	bash
ab123456	4800	4711	0	10:05	pts/0	00:00:00	ps -ef

ps, ps -ef	moje / wszystkie procesy
top, htop	na żywo (q = wyjście)
kill PID	poproś o zakończenie
kill -9 PID	zabij bezwarunkowo
echo \$\$	PID bieżącej powłoki

- **PID** — numer procesu, **PPID** — numer rodzica.
- program & — uruchom w tle.
- Ctrl+Z wstrzymuje; bg / fg wznowia w tle / na pierwszym planie; jobs — lista.

Zmienne

```
licznik=60          # bez spacji!  
napis="Ala ma kota"  
echo $licznik  
echo "${napis}!"    # {} = koniec nazwy  
unset napis  
export EDYTOR=nano
```

- **Zmienne powłoki** — lokalne dla tej powłoki.
- **Zmienne środowiska** (export) — dziedziczą je uruchamiane programy i skrypty.
- set — wszystkie zmienne, printenv — tylko środowiska.

Ważne zmienne środowiska:

\$USER	nazwa użytkownika
\$HOME	katalog domowy
\$HOSTNAME	nazwa komputera
\$SHELL	używana powłoka
\$PWD	katalog bieżący
\$PATH	gdzie szukać programów
\$PS1	wygląd znaku zachęty

Cudzysłowy i podstawianie poleceń

```
echo pwd                # pwd
echo $(pwd)             # /dmj/2026/ab123456  -- wynik polecenia
echo `pwd`              # to samo, starsza składnia
echo "Jestem w $PWD"    # Jestem w /dmj/2026/ab123456
echo 'Jestem w $PWD'    # Jestem w $PWD
n=$(ls | wc -l); echo "Plików: $n"
```

"..."	tekst; \$zmienna i \$(...) są rozwijane ; spacje zostają częścią napisu
'...'	tekst dosłownie , nic nie jest rozwijane
\$(...), `...`	wykonaj polecenie i wstaw jego wynik

Pliki startowe i aliasy

~/.bash_profile	powłoka logowania (np. ssh)
~/.bashrc	każdy nowy terminal
~/.bash_logout	przy wylogowaniu

- Zwykle .bash_profile zawiera `source ~/.bashrc` — wtedy ustawienia wpisujemy tylko do .bashrc.
- Zmiany działają w **nowych** terminalach; w bieżącym: `source ~/.bashrc`.

alias — nowa nazwa polecenia:

```
alias                # lista aliasów
alias dir='ls -la'
alias rm='rm -i'
unalias dir
```

Znak zachęty:

```
PS1='${USER}@${HOSTNAME}:\w$ '
```

Alias zdefiniowany w terminalu znika po jego zamknięciu.

Pierwszy skrypt

Plik witaj.sh:

```
#!/bin/bash
# mój pierwszy skrypt
echo "Witaj $USER"
echo -n "Dziś jest "
date
```

- Każda linia to osobne polecenie (jak wpisane w terminalu).
- `#!/bin/bash` (*shebang*): uruchamiaj w bash, niezależnie od powłoki, w której pracujemy.

Trzy sposoby uruchomienia:

```
bash witaj.sh
chmod +x witaj.sh; ./witaj.sh
source witaj.sh      # albo: . witaj.sh
```

- Dwa pierwsze: skrypt działa w **nowej** powłoce (potomnej); jego cd, zmienne, aliasy znikają po zakończeniu.
- source: polecenia wykonuje **bieżąca** powłoka — np. cd w skrypcie zmienia katalog w terminalu. Tak wczytujemy .bashrc.
- ./ jest potrzebne, bo katalogu bieżącego nie ma w \$PATH.

Argumenty, kod wyjścia, wczytywanie danych

```
./skrypt.sh ala ma kota
```

\$0	nazwa skryptu
\$1 ... \$9	kolejne argumenty (\$1=ala)
\$#	liczba argumentów (3)
*, "\$@"	wszystkie argumenty
\$?	kod wyjścia ostatniego polecenia

- Kod wyjścia **0** = sukces, **inny** = błąd.
- `exit 1` kończy skrypt z kodem 1.

```
echo "Podaj liczbę:"  
read a  
echo "Podwojona: $((2 * a))"
```

```
ls nie_ma_takiego 2> /dev/null  
echo $?          # 2 -- błąd
```

- `read zm` wczytuje linię z `stdin` do zmiennej.
- `$(( ... ))` — arytmetyka na liczbach całkowitych: `+` `-` `*` `/` `%`.

Pisanie do pliku ze skryptu: *here-document*

```
cat << KONIEC >> ~/.bashrc
alias rm='rm -i'
alias cp='cp -i'
KONIEC
```

- Wszystko między << KONIEC a linią KONIEC trafia na stdin polecenia.
- KONIEC można zastąpić dowolnym słowem; linia zamykająca musi zaczynać się od początku linii.
- Zmienne są rozwijane; << 'KONIEC' wyłącza to.

Krótsze teksty — echo:

```
echo "linia 1" > plik.txt
echo "linia 2" >> plik.txt
echo -n "bez nowej linii"
echo -e "a\tb"    # \t = tabulator
```

Instrukcja warunkowa

```
if warunek; then
    polecenia
elif warunek2; then
    polecenia
else
    polecenia
fi
```

- **Warunek to polecenie:** kod 0 = prawda, inny = fałsz.
- Najczęściej test albo równoważne [...].
- then w nowej linii — wtedy bez ;.

```
x=15; y=17
if [ $x -eq $y ]; then
    echo "x = y"
else
    echo "x != y"
fi

if grep -q root /etc/passwd; then
    echo "jest root"
fi
```

Spacje są ważne

[\$x -eq \$y] — [to polecenie, a \$x, -eq, ...,] to jego argumenty. [\$x -eq \$y] nie zadziała.

Warunki: test / []

Liczby

-eq	=
-ne	≠
-gt	>
-ge	≥
-lt	<
-le	≤

Napisy

"\$a" = "\$b"	równe
"\$a" != "\$b"	różne
-z "\$a"	puste
-n "\$a"	niepuste

Pliki

-e p	istnieje
-f p	zwykły plik
-d p	katalog
-s p	niepusty
-r, -w, -x	prawa
! -e p	nie istnieje

```
if [ -z "$1" ]; then
    echo "Użycie: $0 katalog"
    exit 1
fi
```

- Zmienne w warunkach bierzemy w **cudzysłów**: pusta zmienna lub napis ze spacją inaczej psuje składnię.

Operatory logiczne

<code>A && B</code>	B tylko gdy A się udało (AND)
<code>A B</code>	B tylko gdy A się nie udało (OR)
<code>! A</code>	negacja

```
if [ -z "$1" ] || [ -z "$2" ]; then
    echo "Podaj dwa argumenty"
    exit 1
fi
```

Krótkie warunki bez if:

```
[ -d wyniki ] || mkdir wyniki
cd tkr && ls
mkdir -p a/b/c      # nie narzeka,
                   # gdy już istnieje
```

Pętla for

```
for i in 1 3 5 7 9; do
    echo $i
done

for i in $(seq 1 10); do
    echo $i
done

for plik in *.txt; do
    echo "$plik: $(wc -l < "$plik")"
done
```

Jak w C:

```
max=15
for ((i = 1; i <= max; i++)); do
    echo $i
done
```

Po wszystkich argumentach skryptu:

```
for arg; do          # = for arg in "$@"
    echo "$arg"
done
```

Pętla while i wybór case

```
n=1
while [ $n -le 5 ]; do
    echo $n
    n=$((n + 1))
done

# czytanie pliku linia po linii
while read linia; do
    echo ">> $linia"
done < plik.txt
```

```
read -p "Działanie: " op
case "$op" in
    "+") echo $((a + b)) ;;
    "-") echo $((a - b)) ;;
    "*") echo $((a * b)) ;;
    "/" ) echo $((a / b)) ;;
    *)   echo "Nieznana operacja" ;;
esac
```

*) — wszystko inne; ;; kończy gałąź.

Przykład: skrypt archiwizujący

```
#!/bin/bash
# Użycie: ./archiwum.sh katalog
kat=$1
if [ -z "$kat" ]; then
    echo "Użycie: $0 katalog" >&2
    exit 1
fi
if [ ! -d "$kat" ]; then
    echo "Nie ma katalogu $kat" >&2
    exit 1
fi
cel="${kat}_${date +%Y%m%d}.tar.gz"
tar czf "$cel" "$kat" && echo "Zapisano $cel"
```

- Najpierw sprawdzamy argumenty, potem działamy.
- Komunikaty o błędach na stderr: >&2.
- `$(date +%Y%m%d)` — dzisiejsza data, np. 20261006.
- Skrypt działa z dowolnego katalogu, bo nie zakłada, gdzie leży.

Gdy coś nie działa

- Czytaj komunikat błędu — zwykle mówi dokładnie, co jest nie tak.
- `bash -x skrypt.sh` — wypisuje każde polecenie przed wykonaniem.
- Testuj polecenia w terminalu, zanim wpiszesz je do skryptu.
- `man polecenie, polecenie --help`.

Typowe błędy:

- spacje wokół `=`: `x = 5` (źle), `x=5` (dobrze);
- brak spacji w `[ ]`;
- zmienna bez cudzysłowu zawierająca spację;
- brak `chmod +x` albo `./`;
- plik zapisany w Windows (końce linii `\r\n`): `dos2unix plik`.

Podsumowanie

- Polecenia: `ls cd pwd mkdir cp mv rm chmod tar`.
- Oglądanie: `cat less head tail wc du`.
- Szukanie: `find` (nazwy), `grep` (treść).
- Strumienie: `> >> < 2> | i /dev/null`.
- Filtry: `sort uniq cut tee`.
- Zmienne, `export`, `$(...)`, cudzysłowy.
- `.bashrc`, `alias`.
- Skrypty: `#!/bin/bash`, `$1 $# $?`, `read`, `$((...))`.
- Sterowanie: `if [ ]`, `for`, `while`, `case`.

Ćwiczenia

Trzy zestawy zadań (zad. 1–16) — od operacji na plikach, przez potoki, do skryptów z argumentami i warunkami.