Conditions - case

Equivalent of `if/then/elif/elif/.../else/fi` statements chain

- Can use shell pattern matching (e.g. `*`)
- Use `|` for OR of matches
- On match the sequence of commands is run till `;;`
- `*)` typically used for safety `else` with an error message that there was no match

```
case $varname in
  pattern1)
    command1
    ;;
  pattern2|pattern3|pattern4)
    command2
    ;;
  *)
    command_error_no_match
esac
```

Loops - while/until/do/done

Keep looping (un)till EXPRESSION is valid

- Assuming the arguments in the EXPRESSION are changing during the sequence of commands in the loop, thus allowing the loop to stop at some point
- Can immediately stop the loop with `break` command
- Can immediately jump to next iteration with `continue` command

While

Stop looping if EXPRESSION becomes false

```
while [ EXPRESSION ]
do
   commands
   if [ ... ] ; then break ; fi     # alternative way to stop the loop
done
```

Until

Stop looping when EXPRESSION becomes true

```
until [ EXPRESSION ]
do
   commands
   if [ ... ] ; then break ; fi     # alternative way to stop the loop
done
```

For cycle

Loop over predefined list of items

- The list of items to cycle over is space-separated
- Can immediately stop the loop with `break` command
- Can immediately jump to next iteration with `continue` command
- `seq 1 100` to generate list of indexes from 1 to 100

```
for var_i in item1 item2 item3
do
   commands
   if [ ... ] ; then break ; fi     # possible way to stop the loop prematurelly
done
```

Space separation in list

- Potentially dangerous when list contains items with space, e.g. weird filenames
- For files use `find` command instead of `for` cycle
- Or replace spaces by a defined string and inside the loop revert this replacement:

```
# Would not work for files with space
for i in $(ls -1) ; do
  echo $i
done

# Works:
for ii in $(ls -1 | sed 's, ,__mezera__,g') ; do
  i=$(echo $ii | sed 's,__mezera__, ,g')
  echo $i
done
# Works
find . -maxdepth 1 -name '*' -exec echo {} \;
# Works
find . -maxdepth 1 -name 'a*' | while read i ; do echo $i ; done
```

For cycle

Loop over predefined list of items - cont'd

- The list of items to cycle over can be defined alternatively like:

```
for i in {1..5};do echo $i ;done
# from BASH v4.0+,  {START..END..INCREMENT} syntax
for in {0..10..2};do echo $i ;done
# control the width of the loop item:
for i in {001..500};do echo $i ;done
# or combining with other character and multiple ranges
for i in a{001..500} {700..999};do echo $i ;done

## The C-style Bash for loop
for (( initializer; condition; step ))
for (( c=1; c<=5; c++ ));do echo $c ;done
```

Functions

Similar behaviour as in other programming languages

- Mostly to help organization/readabilty of the code
- Accept parameters, treated in similar way as input parameters of scripts (i.e. $1, $2, etc.)
- Output transfered via `echo` command or e.g. my modifying a "global" variable

```
x=0

myfunc() {
  for i in $@ ; do
    echo $i
  done
  x=1
}

echo $x
myfunc a bb cc 123
echo $x
x=0
str=‘myfunc dd ee‘ # x is not changed, myfunc is run in separate shell !
echo $str
echo $x
```

Automatic script options

Use of `getopt` command

- Colon `:` after option letter specifies that the option is expecting an argument

```
while getopts 'ha:' OPTION; do
  case "$OPTION" in
    h)
      echo "Option h (does not expect argument)"
      ;;

    a)
      echo "Option a with value \"$OPTARG\""
      ;;

    ?)
      echo "script usage: $(basename $0) [-h] [-a somevalue]" >&2
      exit 1
      ;;
  esac
done
shift "$(($OPTIND -1))"

echo "Remaining input arguments: $@"
```

Scripts - exercises

- Exercise 1: How to compare floating-point numbers ? Hint `bc -l`, `python -c ... exit,print`
- Exercise 2: Loop through all links in current directory (and sub-directories), check the file really exists (link is valid)
- Exercise 3: Store script input parameters into variables array. Iteratively destroy input parameters one by one and print the remaining on the screen (try all `for`, `while` and `until` loops)

Scripts - exercises

- Exercise 1: How to compare floating-point numbers ? Hint `bc -l`, `python -c ... exit,print`
- Exercise 2: Loop through all links in current directory (and sub-directories), check the file really exists (link is valid)
- Exercise 3: Store script input parameters into variables array. Iteratively destroy input parameters one by one and print the remaining on the screen (try all `for`, `while` and `until` loops)
- Exercise 4: For cycle to generate N random numbers (N=1000 if no argument passed to the script) and print the highest value. Hint: `$RANDOM`.
- Exercise 5: Select random 500 lines from mcData.txt (make sure the lines do not repeat)
- Exercise 6: Loop through archives `backup*`, search for files named `Invariant_masses.txt`, join their content with `mcData.txt` and remove duplicated lines
- Exercise 7: Batch analysis: script triggering a computation jobs
  - Job = generate 100 random numbers with given seed in `rnd.txt`, sleep 1 sec between the generation
  - Run max. 5 jobs in parallel
  - Allow the script to run more than once without breaking the rule above
  - Hint: use flag-files or `ps axuf` to find out which jobs are running, which are finished

Scripts running after logout

nohup

- Most simple way to keep process running after logout (or killing mother terminal)
- Syntax: `nohup command arguments`
- Output goes to `nohup.out` file

screen

- More complex system, behaving as a virtual terminal, allowing to:
  - Detach and re-attach to running session
  - After re-attaching one can see the output of the session
  - Works better on remote machines with complex authentication
  - Can name sessions
  - `screen` allows to send command to a running detached session
- `screen` to start a session
  - *CTRL-a d* to detach from session
  - `screen -list` to list sessions (either attached or detached)
  - `screen -r` to attach to a sessions

tmux

- Similar functionality to `screen`, but more actively developed
- `tmux` to start a session
  - *CTRL-b d* to detach from sessions
  - *tmux ls* to list sessions
  - *tmux attach* to attach

Crontab - Reminder

CRON system:

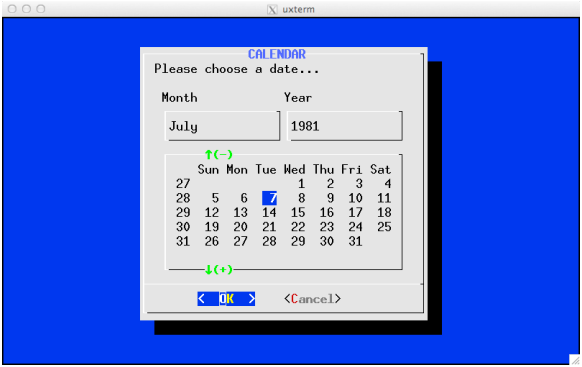
- /etc/crontab: basic file to run tasks per hour/day/week/month
- /etc/cron.hourly
- /etc/cron.daily
- /etc/cron.weekly
- /etc/cron.monthly
- /etc/cron.d: more complicated rules

```
# /etc/cron.d/renew_prak0x: crontab entries for reweval of the prak0x user home directories
# Execute only during the period of the exercises (01.Oct - 20.Jan)
# TODO ?: Add entry in between day in case of 2 excercises per single day

SHELL=/bin/bash

# m  h   dom   mon              dow  user  command
32   01   *    OCT,NOV,DEC,JAN  SUN  root  /home/prak_template/bin/reboot.cron.sh
# NO!!! (studenti by po rebootu nenasli sva data !)
#@reboot                            root  /home/prak_template/bin/renew_prak0x.cron.sh
12   03   *    OCT,NOV,DEC       *   root  /home/prak_template/bin/renew_prak0x.cron.sh
12   03  1-20  JAN               *   root  /home/prak_template/bin/renew_prak0x.cron.sh
```

Graphical interface to scripts

Programs to easily create simple graphics interfaces:

- Calendar
- File selection
- Forms
- Messages
- Lists
- Progress bars
- Text entry



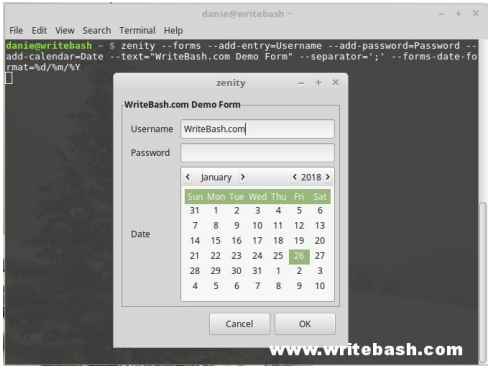
dialog

- Terminal-based graphics
- See number of exampes in `/usr/share/doc/dialog/examples`

Graphical interface to scripts

Programs to easily create simple graphics interfaces:

- Calendar
- File selection
- Forms
- Messages
- Lists
- Progress bars
- Text entry



dialog

- Terminal-based graphics
- See number of exampes in `/usr/share/doc/dialog/examples`

zenity / gdialog

- Graphical windows (GTK)
- See examples at https://help.gnome.org/users/zenity/3.32/