Programmeertalen: Bash

Shell and Scripting Language

Bas Terwijn

Universiteit van Amsterdam

4 februari 2020

Programming paradigms

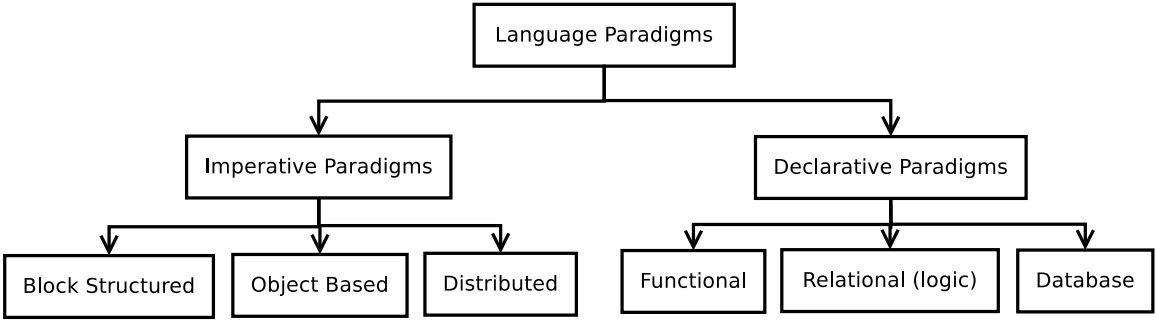
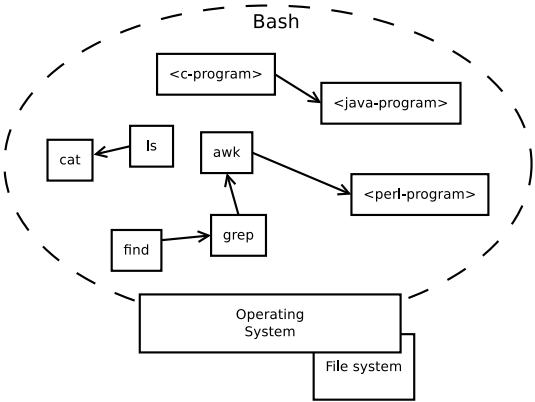


Figure: Programming Languages: Paradigm and Practice, Doris Appleby

Domain specific language



- close to the operating system
- high level, runs other programs to do the work
- changes and streams data from one program to the other
- quick and dirty

Language Features

- type system: string based
- pipelined: Yes
- higher order function: Yes
- concurrent: Yes

Goal of this week

- Practical use of:
  - the Shell
  - Regular Expressions (regex)
  - Makefiles
  - Bash Scripting

Bash alternatives

- Bash (Bourne Again Shell), standard shell on Ubuntu
- many others: sh, csh, tsh, ksh, ash, dash, zsh, fish, ...

Basic shell commands

```
$ pwd
/home/bterwijn
$ mkdir bash
$ cd bash
$ pwd
/home/bterwijn/bash
$ ls
$ echo "hello world"
hello world
$ echo "hello world" > test.txt
$ ls
test.txt
$ cat test.txt
hello world
```

Basic shell commands

```
$ cp test.txt test2.txt
$ mkdir dir dir2
$ ls
dir2  dir   test2.txt  test.txt
$ rm test.txt
$ rm dir2
rm: cannot remove 'dir2': Is a directory
$ rm -r dir2  # or use 'rmdir' to remove empty directory
$ ls
dir test2.txt
$ ls *?[0-9].txt # *:zero-or-more ?:single []:range
test2.txt
```

Documentation

```
$ man ls  # '/':search, n:next: N:previous
LS(1)                          User Commands                          LS

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by defaul
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  spe
       fied.

       Mandatory  arguments  to  long  options are mandatory for short opti
       too.
```

Documentation

```
$ ls --help  # or often: '-h' '-help'
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
  -b, --escape               print C-style escapes for nongraphic character
      --block-size=SIZE      scale sizes by SIZE before printing them; e.g.
                               '--block-size=M' prints sizes in units of
                               1,048,576 bytes; see SIZE format below
  -B, --ignore-backups       do not list implied entries ending with ~
  -c                         with -lt: sort by, and show, ctime (time of la
                               modification of file status information);
```

File permissions

```
$ ls -al
total 16
drwxr-xr-x  3 bterwijn bterwijn 4096 feb  2 16:58 .
drwxr-xr-x 68 bterwijn bterwijn 4096 feb  2 16:33 ..
drwxr-xr-x  2 bterwijn bterwijn 4096 feb  2 16:57 dir
-rw-r--r--  1 bterwijn bterwijn   12 feb  2 16:35 test2.txt
            | |        |           | |            |
            | |        |           | |            +- file name
            | |        |           | +- last modification time
            | |        |           +- size in bytes
            | |        +- group the file belongs to
            | +- owner the file belongs to
            +- number of directories
```

File permissions

```
$ ls -al
total 16
drwxr-xr-x  3 bterwijn bterwijn 4096 feb  2 16:58 .
drwxr-xr-x 68 bterwijn bterwijn 4096 feb  2 16:33 ..
drwxr-xr-x  2 bterwijn bterwijn 4096 feb  2 16:57 dir
-rw-r--r--  1 bterwijn bterwijn   12 feb  2 16:35 test2.txt
||  |  |
||  |  +- read/write/execute bits for 'others'
||  +- read/write/execute bits for 'group'
|+- read/write/execute bits for 'owner'
+ d:directory, l:link, -:file
```

File permissions

```
$ chmod o-rx dir        # remove read and execute rights for 'others'
$ chmod u+x test2.txt   # add execute rights for 'user'(=owner)
$ chown bterwijn.othergroup test2.txt  # change the owner and group
$ ln -s test2.txt link.txt             # add a link to test2.txt
$ ls -l
drwxr-x--- 2 bterwijn bterwijn  4096 feb  2 16:57 dir
lrwxrwxrwx 1 bterwijn bterwijn     9 feb  2 17:25 link.txt -> test2.txt
-rwxr--r-- 1 bterwijn othergroup  12 feb  2 16:35 test2.txt
$ cat link.txt
hello world
$ sudo rm -r dir #run as other user (default:'root'), requires password
```

Read files

command	***effect***
cat <filename>	print file to stdout
more <filename>	scroll through file, Enter:scroll-line Space:scroll-page q:end h:help
less <filename>	advanced version of "more", '/':search n:next N:previous
head -n 5 <filename>	first 5 lines
tail -n 5 <filename>	last 5 lines
tail -f /var/log/syslog	keep reading file as lines are added
wc <filename>	count nr of: newlines, words, bytes
grep <string><filename>	print lines that contain <string>

Read files

test.txt

```
dit is een test file, hello
Hello World 12345 AA BB CC
laatste regel
```

```
$ wc test.txt
 3 14 86 test.txt
$ grep hello test.txt
dit is een test file, hello
$ grep -Hni "hello" test.txt # H:filename n:regelnr i:case-insensitive
test.txt:1:dit is een test file, hello
test.txt:2:Hello World 12345 AA BB CC
```

Redirect out stream

- System.out.println("hello"); prints to stdout
- System.err.println("hello"); prints to stderr

```
$ echo "Hello World 1" >   t.txt   # stdout to file, overwrites t.txt
$ ls non-exist1        2>  t.txt   # stderr to file, overwrites t.txt
$ echo "Hello World 2" >>  t.txt   # stdout to file, append to t.txt
$ ls non-exist2        2>> t.txt   # stderr to file, append t.txt
$ cat t.txt                        # file to stdout
ls: cannot access 'non-exist1': No such file or directory
Hello World 2
ls: cannot access 'non-exist2': No such file or directory
$ cat t.txt      2>/dev/null 1>&2  # file to stderr to /dev/null
```

Redirect input stream

```
$ grep hello      # now typing "123 hello abc"<Return><Ctrl-d>
123 hello abc
$ grep hello < t.txt                      # file to stdin
$ grep hello < <(echo "123 hello abc")  # stdout to stdin
123 hello abc
```

Pipes

```
$ echo "1 hello there" >  t.txt         # stdout to file
$ echo "2 hello world" >> t.txt         # stdout to file
$ cat t.txt | grep hello | grep world  # stdout to stdin ... to stdout
2 hello world
```

String manipulation

test.txt

```
hallo: 123 abc eee
world: 101 123456789 klawmf
```

```
$ cat test.txt | tr '1' '#'          # replace single character
hallo: #23 abc eee
world: #0# #23456789 klawmf
$ cat test.txt | sed 's/123/XYZ/g'   # replace words
hallo: XYZ abc eee
world: 101 XYZ456789 klawmf
$ cat test.txt | cut -d':' -f1       # cut at ':' and print field 1
hello
world
```

String manipulation

test.txt

```
hallo: 123 abc eee
world: 101 123456789 klawmf
```

```
$ cat test.txt | awk '{print "["$2"]-["$4"]"}'
[123]-[eee]                    # print column 2 and 4 in square brackets
[101]-[klawmf]
$ cat test.txt | awk '{print NR*10,$0}'
10 hallo: 123 abc eee          # add <line-number*10> to each line
20 world: 101 123456789 klawmf
$ cat test.txt | awk '{total += $2} END {print total}'
224                            # compute total of column 2
```

String manipulation

names1.txt

```
Emma
Lucas
Emma
```

names2.txt

```
Nathan
Louis
Lucas
```

```
$ sort names1.txt                # sort alphabetically
Emma
Emma
Lucas
$ sort names1.txt | uniq         # sort and remove duplicates
Emma
Lucas
$ sort names1.txt | uniq | wc -l  # count unique items
2
```

String manipulation

names1.txt

```
Emma
Lucas
Emma
```

names2.txt

```
Nathan
Louis
Lucas
```

```
$ sort names1.txt | uniq > n1.txt    # sort and remove duplicates
$ sort names2.txt | uniq > n2.txt    # sort and remove duplicates
$ sort n1.txt n2.txt | uniq -d       # set intersection
Lucas
```

Regular Expressions

\d	digit
\D	not a digit
\w	word character (a-z,A-Z,0-9,_)
\W	not a word character
\s	whitespace (space,tab,new line)
\S	not a whitespace
\b	word boundry
\B	not a word boundry
^	beginning of string
$	end of string

[ ]	character range: [xyz] [0-9]
[^]	not in character range
.	any character
?	optional
*	zero or more times
*?	zero or more times not greedy
+	one or more times
+?	one or more times not greedy
()	capture group
(?1)	subroutine

Regular Expressions

\d	digit
\D	not a digit
\w	word character (a-z,A-Z,0-9,_)
\W	not a word character
\s	whitespace (space,tab,new line)
\S	not a whitespace

```
$ echo "test 123 abcDEF" | perl -pe 's/\D/[]/sg'
[][][][][]123[][][][][][][][]
$ echo "test 123 abcDEF" | perl -pe 's/\s/[]/sg'
test[]123[]abcDEF[]
```

Regular Expressions

\b	word boundry
\B	not a word boundry
^	beginning of string
$	end of string

```
$ echo "test 123 abcDEF" | perl -pe 's/\D\b/[]/sg'
tes[][]123[]abcDE[]
$ echo "test 123 abcDEF" | perl -pe 's/\w$/[]/sg'
test 123 abcDE[]
```

Regular Expressions

[ ]	character range: [xyz] [0-9]
[^]	not in character range
.	any character
?	optional

```
$ echo "test 123 abcDEF" | perl -pe 's/[^a-z]/[]/sg'
test[][][][][]abc[][][][]
$ echo "test 123 abcDEF" | perl -pe 's/ab.DE?/[]/sg'
test 123 []F
```

Regular Expressions

*	zero or more times
*?	zero or more times not greedy
+	one or more times
+?	one or more times not greedy

```
$ echo "aaaaabbbccccbbbbbcccc" | perl -pe 's/ab*c/[]/sg'
aaaa[]cccbbbbbcccc
$ echo "aaaaabbbccccbbbbbcccc" | perl -pe 's/a.+c/[]/sg'
[]
$ echo "aaaaabbbccccbbbbbcccc" | perl -pe 's/a.+?c/[]/sg'
[]cccbbbbbcccc
```

Regular Expressions

() capture group
$1 value of first group
\1 match with value of first group
(?1) copy first group

```
$ echo " AxxB AxyB AyyB " | perl -pe 's/A(.*?)B/A$1-$1-$1B/sg'
 Axx-xx-xxB Axy-xy-xyB Ayy-yy-yyB
$ echo " AxxB AxyB AyyB " | perl -pe 's/A(.)\1B/XX/sg'
 XX AxyB XX
$ echo " AxxB AxyB AyyB " | perl -pe 's/(A.*?B)\s*(?1)/XX/sg'
 XX AyyB
```

Search files

names1.txt

```
./A
./A/file.dat
./A/a1
./A/a1/file1.txt
./A/a2
./A/a2/file2.txt
```

```
$ find ./ -name '*.txt'
./A/a1/file1.txt
./A/a2/file2.txt
$ find -type d -ctime -24 -name 'a*' # directory, changed in last 24h
./A/a1
./A/a2
```

Search files

names1.txt

```
./A
./A/file.dat
./A/a1
./A/a1/file1.txt
./A/a2
./A/a2/file2.txt
```

```
$ find -name '*.txt' -exec grep -Hn hello {} \;
./A/a1/file1.txt:15:  hello world
```

Makefile

```
$ ls
Makefile test.txt

$ make test.pdf
enscript test.txt -o test.ps
ps2pdf test.ps > test.pdf

$ make    # 'all' is default
make: Nothing to be done for 'all'.

$ ls
Makefile test.txt test.pdf

$ make clean
rm -f test.pdf
```

Makefile

```
TEXT=$(wildcard *.txt)
PDFS=$(patsubst %.txt,%.pdf,$(TEXT))

all: $(PDFS)

%.ps: %.txt
        enscript $< -o $@

%.pdf: %.ps
        ps2pdf $< > $@

clean:
        rm -f $(PDFS)
```

Install programs

```
$ apt search datamash          # search for example package "datamash"
Sorting... Done                # on Mac use "homebrew" package manager
Full Text Search... Done
datamash/bionic,now 1.2.0-1 amd64 [installed]
  statistics tool for command-line interface

$ apt show datamash
... package description ...

$ sudo apt remove datamash     # remove package

$ sudo apt install datamash    # install package, downloads a .deb file

$ snap search datamash         # same packages for any Linux distribution
```

Install programs

```
$ sudo dpkg -i datamash.deb    # manually install a .deb file
$ sudo dpkg -r datamash        # manually remove package

$ dpkg -L datamash             # list files installed by package
/usr
/usr/bin
/usr/bin/datamash
...

$ dpkg -S /usr/bin/datamash    # search what package installed file
datamash
```

Bash Scripting, numbers

```
let "count = 14 % 5";
echo $count
count=$(( count * 10 ))
echo $count
echo $(( count / 9.0 ))
```

```
4     # a number is just a string of digits
40
4     # no floating point numbers
```

Bash Scripting, array

```
array=(zero one two three)
array+=('four')
echo "Array size: ${#array[*]}"

echo -n "Array items: "
for item in ${array[*]}; do echo -n "$item " done

echo -en "\nArray index&items: "
for index in ${!array[*]}; do echo -n "$index:${array[$index]} " done
```

```
Array size: 5
Array items: zero one two three four
Array items: 0:zero 1:one 2:two 3:three 4:four
```

Bash Scripting, associative array

```
declare -A MYMAP=( [foo]=bar [1]=one ) # init map
echo "foo: ${MYMAP[foo]}"
MYMAP[baz]=new_value                   # add to map
echo "baz: ${MYMAP[baz]}"
[[ ${MYMAP[1]+_} ]] && echo "found" || echo "Not found"
[[ ${MYMAP[2]+_} ]] && echo "found" || echo "Not found"
unset MYMAP[foo]                       # remove from map
for i in "${!MYMAP[@]}"; do echo -n " $i:${MYMAP[$i]} "; done # loop
```

```
foo: bar
baz: new_value
found
Not found
 baz:new_value  1:one
```

Bash Scripting, conditions

```
number="42"

if [[ "$number" = "42" ]]; then  # whitespace are important!
  echo "expression evaluated as true"
else
  echo "expression evaluated as false"
fi

[[ "$number" = "42" ]] && echo "expression evaluated as true"

case $number in
42) echo "equal to 42" ;;
*)  echo "not equal to 42"
esac
```

Bash Scripting, conditions

[[string = string]]

=	equal
!=	not equal
<	smaller alphabetically
>	larger alphabetically
-n	not empty
-z	empty

[[number -eq number]]

-eq	equal
-ne	not equal
-lt	less than
-gt	greater than
-le	less or equal
-ge	greater or equal

```
if [[ "$n" -gt "0" ]] && [[ "$n" -lt "100" ]] ||
   [[ "$n" -eq "1000" ]]; then
  echo "between 0 and 100 or equal to 1000"
fi
```

Bash Scripting, control flow

```
for i in $( ls ); do
  echo "file: $i"
done

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

for ((i=0; i<10; i++ ))
do
   echo -n "$i "
done
```

Bash Scripting, control flow

```
let "i = 0";
while [[ $i -lt 10 ]]; do
  echo -n "$i "
  let i+=1
done

let "i = 0";
while [[ true ]]; do
  echo -n "$i "
  let i+=1
  [[ $i -ge 10 ]] && break  # no do-while, use break
done
```

Bash Scripting, IO

```
while read -r line; do
    echo "read line: $line"
done                # read from stdin

filename="myFile.txt"
while read -r line; do
    echo "read line: $line"
done < "$filename"  # read from file
```

Bash Scripting, arguments

```
echo "nr-arguments: $#"
echo "first 4 arguments: $1 $2 $3 $4"
for arg in "$@"; do
    echo "$arg"
done
```

Bash Scripting, functions

```
function divide {   # divide first argument by second
  if [[ $2 -eq "0" ]]; then
    return 1   # error state
  else
    echo "$(( $1 / $2 ))"    # the result is printed!
    return 0   # ok state
  fi
}
result=$( divide 10 2 )
[[ "$?" -ne "0" ]] && echo "error" || echo "result: $result"
result=$( divide 10 0 )
[[ "$?" -ne "0" ]] && echo "error" || echo "result: $result"
```

```
result: 5
error
```

Bash Scripting, higher order functions

```
function divide {   # divide first argument by second
  if [[ $2 -eq "0" ]]; then
    return 1   # error state
  else
    echo "$(( $1 / $2 ))"    # the result is printed!
    return 0   # ok state
  fi
}

function do_arithmetic { # apply function $1 to $2 $3
  $1 $2 $3
  return $?
}

do_arithmetic divide 10 2
```

Bash Scripting, scope

```
X="old"
Y="old"
function myFunction {
  local X
  X="new"
  Y="new"
  echo "myFunction: X:$X  Y:$Y"
}
myFunction
echo "main:       X:$X  Y:$Y"
```

```
myFunction: X:new  Y:new
main:       X:old  Y:new
```

Process management

```
$ sleep 100                 # blocks for 100 seconds
<Ctrl-z>                    # suspend (or <Ctrl-c> to kill)
$ bg                        # moves 'sleep' to run in background
$ ps aux | grep sleep       # snapshot of current processes
bterwijn 11065  0.0 ...     22:45   0:00 sleep 100
bterwijn 11080  0.0 ...     22:45   0:00 grep --color=auto sleep
$ fg                        # moves 'sleep' to foreground, blocks again
<Ctrl-z>                    # suspend
$ bg                        # moves 'sleep' to run background again
$ kill 11065                # kill the process
$ ps aux | grep sleep       # snapshot of current processes
bterwijn 11088  0.0 ...     22:46   0:00 grep --color=auto sleep
$ sleep 100 &               # run in background
$ top                       # show current processes sorted by CPU usage
```

Inter-process communication

```
pipe=/tmp/testpipe
rm -f $pipe
mkfifo $pipe
while true; do
  if read line <$pipe; then
    [[ "$line" = 'q' ]] && break
    echo $line
  fi
done
echo "Reader exiting"
```

```
pipe=/tmp/testpipe
if [[ ! -p $pipe ]]; then
    echo "Reader not running"
    exit 1
fi
if [[ "$1" ]]; then
    echo "$1" >$pipe
else
    echo "Hello from $$" >$pipe
fi
```

Assignments

- Knapsack: `https://bitbucket.org/bterwijn/knapsack`
  - bitbucket: free private repositories with @uva.nl
- individual: change and analyze Java source code
- team: run knapsack experiments to find good solution