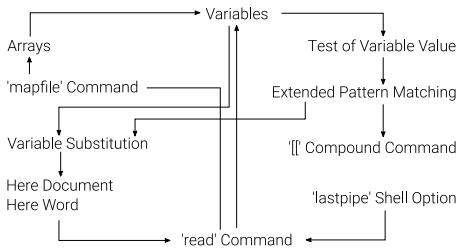
bash

Text Processing



$'string'

Backslash-escaped characters are replaced as specified by ANSI C standard

```
bold=$'\e[1m'
norm=$'\e[0m'
pop=$'\U1f4a9'
```

UNIX → DOS

```
printf "%s\r\n" "${text//$'\n'/$'\r\n'}"
```

```
sed 's/$/\r/' <<<"$text"
```

```
IFS=$'\n'
printf "%s\r\n" $text
```

$(cat *file*) Vs $(< *file*)

```
for ((i=0;i<10000;i++)); do
    : "$( cat /etc/passwd )"
done
```

```
real    0m12.186s
user    0m3.008s
sys     0m8.657s
```

```
for ((i=0;i<10000;i++)); do
    : "$( < /etc/passwd )"
done
```

```
real    0m6.364s
user    0m1.442s
sys     0m4.547s
```

```
real    3m40.018s
user    0m53.207s
sys     1m30.623s
```

```
getent passwd \
| gawk -F: '! system("test -d /home/zam/"$1) { print $1 }'
```

```
real    2m40.790s
user    0m45.622s
sys     2m11.267s
```

```
getent passwd \
| while read line
  do
     u=$( echo "$line" | cut -d: -f1 )
     [ -d /home/zam/$u ] && echo $u
  done
```

```
real    0m6.691s
user    0m3.250s
sys     0m2.484s
```

```
IFS=:
getent passwd \
| while read u rest
  do [ -d /home/zam/$u ] && echo $u; done
```

Transaction costs

Why?

Why NOT?

This is the Unix philosophy:

> Write programs that do one thing and do it well.
>
> Write programs to work together.
>
> Write programs to handle text streams, because that is a universal interface.

Doug McIlroy, the inventor of Unix pipes

bash
Text Processing

lukas.barinka@gmail.com

Variables (Named Parameters)

Store / find out value of variable

```
a=5
```
Store value into variable

```
declare -p a
declare -- a="5"
```
Find out value of variable

```
a+=5
declare -p a
declare -- a="55"
```
Add value

Integer type of variables

```
declare -i a
```
Integer attribute

```
a+=5
declare -p a
declare -i a="60"
```
Arithmetic operations with integer variables

```
declare -i b
b=a+1
declare -p b
declare -i b="61"
```

```
b=c=a
declare -p a b c
declare -i a=60
declare -i b=60
declare -- c=60
```

Command Substitution

```
for ((i=0;i<1000;i++)); do
   n=$( printf %03d $i )
done
```
Sub-shell

```
real    0m6.048s
user    0m1.350s
sys     0m4.106s
```

```
real    0m0.170s
user    0m0.170s
sys     0m0.000s
```

```
for ((i=0;i<1000;i++)); do
   printf -v n %03d $i
done
```
Current shell

Here Document & Here Word

```
getent passwd $USER
barinkl:##barinkl:10001:1002:Lukas Barinka zam\
:/home/zam/barinkl:/bin/bash
```

<<word

```
name=$( cut -d: -f5 << END
$( getent passwd $USER )
END
)
```
No substitutions when END is quoted

Here Document

<<<word

```
u=$( getent passwd $USER )

uid=$( echo "$u" | cut -d: -f3 )
name=$( echo "$u" | cut -d: -f5 )
```

Here Word

```
uid=$( cut -d: -f3 <<< "$u" )
name=$( cut -d: -f5 <<< "$u" )

uid=$( cut -d: -f3 <<< "$(getent passwd $USER)" )
name=$( cut -d: -f5 <<< "$(getent passwd $USER)" )

var='     a    b     '

od --endian=big -x <<<$var
... 6120 620a
      a ␣  b ⏎


od --endian=big -x <<<"$var"
... 2020 2020 2061 2020 2062 2020 2020 200a
      ␣  ␣  ␣  ␣  a  ␣  ␣  ␣  b  ␣  ␣  ␣  ␣  ⏎
```

'read' Command

Read one line from standard input

```
IFS=:
read user p u g n h s <<< "$(getent passwd $USER)"
declare -p user p u g n h s
declare -- user="barinkl"
declare -- p="##barinkl"
declare -- u="10001"
declare -- g="1002"
declare -- n="Lukas Barinka zam"
declare -- h="/home/zam/barinkl"
declare -- s="/bin/bash"
```
Characters in IFS are used to split line into words

```
read junk pass junk <<< "$(getent passwd $USER)"
declare -p junk pass junk
declare -- pass="##barinkl"
declare -- junk="10001:1002:Lukas Barinka zam:..."
```

```
read line <<< '   indented   text    '
declare -p line
declare -- line="indented   text"
```
Trim (left+right)

If no names are supplied, line is assigned to variable REPLY

```
read <<< '   indented   text    '
declare -p REPLY
declare -- REPLY="   indented   text    "
```
No trim

Redirection into compound command

```
read r u s <<< $( (time -p sleep 5 ) 2>&1 )
declare -p r u s
declare -- r="real"
declare -- u="5.00"
declare -- s="user 0.00 sys 0.00"
```
1 line with 6 words

```
{ read r; read u; read s; } \
<<< "$( (time -p sleep 5 ) 2>&1 )"
declare -p r u s
declare -- r="real 5.00"
declare -- u="user 0.00"
declare -- s="sys 0.00"
```
3 lines per 2 words

```
while read; do
   cut -d: -f2 <<< "$REPLY"
done <<< "$( getent passwd )"
```
```
real    1m40.755s
user    0m25.508s
sys     1m14.606s
```

```
while read; do
   cut -d: -f2 <<< "$REPLY"
done < <( getent passwd )
```
```
real    1m37.749s
user    0m25.563s
sys     1m15.919s
```

```
while read junk pass junk; do :
done <<< "$( getent passwd )"
```
```
real    0m7.130s
user    0m2.872s
sys     0m0.654s
```

```
while read junk pass junk; do :
done < <( getent passwd )
```
```
real    0m4.248s
user    0m2.394s
sys     0m1.271s
```

'lastpipe' Shell Option

ksh Vs bash Vs bash 4.2+

Each command in a pipeline is executed as a separate process (i.e., in a subshell)

```
(
   shopt -u lastpipe
   echo a | read x
   declare -p x
)
bash: declare: x: not found
```

If lastpipe is set, and job control is not active, shell runs last command of pipeline not executed in the background in current shell environment

```
(
   shopt -s lastpipe
   echo a | read x
   declare -p x
)
declare -- x="a"
```

Test of Variable Value

```
test expression
[ expression ]
```
Built-in command to evaluate LE

```
[[ expression ]]
```
Compound command to evaluate LE/RE

```
case $var in
   pattern) ... ;;
         *) ...;;
esac
```
Compound command to compare value with pattern (or else)

```
shopt -s nocasematch
```

Extended Pattern Matching

If extglob shell option is enabled, extended pattern matching operators are recognized

```
shopt -s extglob
```

Options:

```
nocaseglob
nullglob
dotglob
starglob
globasciiranges
```

anything except → `!(pattern-list)`
zero or one → `?(pattern-list)`
one → `@(pattern-list)`
zero or more → `*(pattern-list)`
one or more → `+(pattern-list)`

```
*.@(jpg|jpeg|gif|png)
file_+([0-9])
```

```
rm $( ls | egrep -v '\.(c|cc|h)$' )
rm !(*.@(c|cc|h))
```

Arrays

(Auto) Indexed array

```
IFS=:
DATE=( $( date +%Y:%m:%d:%T ) )
```
Setup array

```
echo "${DATE[*]}"
2016:03:05:10:00:00
```
Print out array

```
echo "${DATE[@]}"
2016 03 05 10 00 00
```

Read values into array

```
read -a DATE <<< "$( date '+%Y:%m:%d:%T' )"
declare -p DATE
declare -a DATE='([0]="2016" [1]="03" [2]="05"
                  [3]="10" [4]="00" [5]="00")'
```

Associative array

```
declare -A students=()
eval students=(
   $( getent passwd \
      | cut -d: -f1,5 \
      | sed -n 's/^/[/;s/:/]="/;s/ students$/"/p'
   )
)
```
```
real    1m7.418s
user    1m2.722s
sys     0m1.445s
```
```
real    0m6.190s
user    0m5.099s
sys     0m1.558s
```

```
declare -A students=()
IFS=:
while read u p uigi n h s; do
   [[ $n =~ (.*)\ student$ ]] \
   && student[$u]=${BASH_REMATCH[1]}
done
```

'mapfile' Command

```
mapfile -t CONF < file
```
Read lines from standard input into indexed array variable ($MAPFILE = default)
Trim '\n'

mapfile is 2× faster than while loop, but less customizable

```
while read; do CONF+=( "$REPLY" ); done < file
```

'[[' Compound Command & Regular Expressions

```
[[ value =~ ERE ]]
```
Extended regular expression

```
sed -r 's/…(ERE)…(ERE)…/\1\n\2/'
egrep -o ERE
```

```
${BASH_REMATCH[0]}
${BASH_REMATCH[1]}...
```
String matching ERE
Substring matching ( )

```
cd /var/db/pkg; ls -d */* | while read; do
   [[ $REPLY =~ (.*)-(.*)/(.*)-([0-9.]+)(-(.*))? ]] \
   && declare -p BASH_REMATCH
done
… BASH_REMATCH='([0]="x11-misc/shared-mime-info-1.2-r1"
  [1]="x11" [2]="misc" [3]="shared-mime-info"
  [4]="1.2" [5]="-r1" [6]="r1")'
… BASH_REMATCH='([0]="x11-proto/inputproto-2.3"
  [1]="x11" [2]="proto" [3]="inputproto"
  [4]="2.3" [5]="" [6]="")'
```

Variable Substitution

Simple text transformations

```
shopt -s nullglob extglob
for i in !(`date +%Y-%m`*)/
do
   i=${i%/}
   p=${PWD//\//_}
   p=${p#_}
   tar cvzf "${p}_$i.tgz" "$i" && rm -rf "$i"
done
```
```
/var/cache/data/2016-03-05/
var_cache_data_2016-03-05.tgz
```
Removing suffix pattern (shortest)
Pattern substitution (all)
Removing prefix pattern (shortest)

Get last/first item

```
rev | cut -d\  -f1 | rev
awk '{print $NF}'
```

Trim (right)

```
var=${var%%*( )}
var=${var/%*( )}
```
Removing suffix pattern (longest)
Pattern substitution (first)
Anchor (end)

Removing prefix pattern (longest)

```
echo "${var##* }"
echo "${var##*[[:space:]]}"
```
Character class

Substituting filter

```
while read; do
   printf '%s\n' "${REPLY//abc/def}"
done
```
```
sed 's/abc/def/'
sed 's/abc/def/g'
```

Get last character from variable

```
printf '%s\n' "${var: -1}"
```
Substring expansion (offset)

```
tail -c2 <<< "$var" | head -c1
```

Serialize lines of file using ':' character

```
printf -v var %s: $(file)     paste -s -d: file
printf '%s\n' "${var:0:-1}"
printf '%s\n' "${var:0:$(($(#var}-1))}
```

Change case

```
tr a-f A-F
```
offset from right (or length)
offset from left

```
printf '%s\n' "${var^^[a-f]}"
```
pattern (one character)
capitalize (all)