Characters & Expressions

SPECIAL CHARACTERS	have a special meaning, the interpreter is the shell
$	dollar – prompt; indicates readiness of the shell to accept commands (default notation for a regular user)
#	hash 1) prompt; indicates readiness of the shell to accept commands (default notation for the root user) 2) comment in configuration files and scripts; any word placed after this character is ignored by the system, except for a "#!" (shebang) at the beginning of the script followed by the absolute path to the shell or program used to execute the file
/	slash 1) represents the root directory 2) separates directories in the absolute path
.	dot; represents the current (working) directory
..	two dots; represent the parent directory

-	dash
	1) indicates options for the command
	`$ ls -a`
	2) reads data from STDIN
	`$ gzip -cd file.tar.gz | tar -xf -`

--	double dash
	1) indicates long options for the command
	`$ ls --all`
	2) indicates the end of command options (the following string is treated as an argument)
	`$ mkdir -- -dir` | `$ mkdir ./-dir`
	(creates a directory "-dir")

****	backslash (at the end of the line); indicates the line break of the command
>	"greater than" sign – $PS2 (at the beginning of the line); indicates the continuation of the command line
Pipelines:	**sequences of commands separated by one of the following control operators:**
|	pipe; connects the standard output of one command via a pipe to the standard input of another command `$ ls -l | grep "^d" | wc -l` (prints the number of subdirectories in the working directory)
|& | **2>&1 |**	pipe and ampersand; connect the standard output and standard error of one command via a pipe to the standard input of another command
Lists:	**sequences of commands or pipelines separated by one of the following control operators:**
;	semicolon; indicates the end of the command (the same as the new line) `$ cd /tmp; ls -la`
&	ampersand; executes the command in the background in a subshell (useful for running larger jobs; the return code is always zero)
&&	double ampersand; executes the following command only if the previous command ends successfully (returns a zero exit status) `$ mount /mnt/fd && cp -R /mnt/fd /tmp/fd && umount /mnt/fd` (mounts the floppy, copies its contents to "/tmp/fd" and unmounts the floppy)

SPECIAL CHARACTERS	**have a special meaning, the interpreter is the shell**
||	double pipe; executes the following command only if the previous command fails (returns a non-zero exit status) `$ [ $? -eq 0 ] || echo $?` (prints the return code of the previous command if it failed)
Compound commands:	**groups of commands that are treated as a single unit**
(<list>**)**	parentheses; group commands (commands are executed in a subshell) `$ (a; b) & c &` (commands "a" and "b" are executed sequentially in the background, i.e., one by one, command "c" is executed in parallel)

{ <list>**;}**	braces; group commands (commands are executed in the current shell)
	`$ { a; b;} & c &`
	(commands "a" and "b" are executed sequentially in the background, i.e., one by one, command "c" is executed in parallel)
	`$ cd ~/.ssh 2> /dev/null || { mkdir ~/.ssh; cd ~/.ssh;}`
	(changes the current directory to the specified directory, if it does not exist, it is created first)

PARAMETERS	**represent entities that store specific values**
Positional parameters:	**numeric parameters inside a shell script representing the arguments passed to the script on the command line in the corresponding order**
<n> | **{**<n>**}**	numeric parameter starting from 1
Special parameters:	**parameters whose contents are read-only**
<parameter> | **{**<parameter>**}**	**#** inside the script represents the total number of command-line arguments passed to the script, * or **@** inside the script represents all the command-line arguments passed to the script (if the expansion occurs within double quotes, "$*" expands to a single word with each positional parameter separated by a space, whereas "$@" expands each positional parameter to a separate word), **0** inside the script represents the name of the shell script, on the command line it represents the name of the current shell, **$** represents the PID of the current shell, **?** represents the return code of the last foreground process, **!** represents the PID of the last background process, _ represents the last argument of the last command executed, **-** represents the current shell options
Variables:	**"storage" of information, consist of the name and assigned value**

<variable>**=**<value>	definition of a local variable (remains defined in the shell until its termination)
	`$ xy="day"`
	(defines a variable "xy" and assigns it the value "day")
	`$ working_directory=$(pwd)`
	(assigns a variable the value of the command output)
	`$ PATH="$PATH:~/scripts"`
	(adds the "scripts" directory to the "PATH" variable)
	`$ PS1="[\u@\h \W]\$ "`
	(sets the user's prompt; the "PS1" variable can be defined as follows: **\u** = user, **\h** = hostname, **\W** = working directory)

Arrays:	**data structures that allow multiple values to be stored in a single variable**

SPECIAL CHARACTERS	**have a special meaning, the interpreter is the shell**

<indexed_array>**=(**<value1> <value2> <value3> ...**)**	definition of an indexed array (ordered list of items); each value in the array is associated with and accessed by a corresponding numeric index, which starts with 0 for the first item, and increments by 1 for each subsequent item in the array; indexed arrays may optionallly be declared before being defined
	`$ colours=("blue" "red" "light green")` | `$ declare -a colours=("blue" "red" "light green")`
	(defines an indexed array "colours" and assigns it three different values)
	`$ colours+=("yellow" "white")`
	(appends additional values to the indexed array)
	`$ colours[3]="black"`
	(assigns the value "black" to the item of the indexed array with an index of 3)

<associative_array>**=(**<key1> <value1> <key2> <value2> <key3> <value3> ...**)** | <associative_array>**=([**<key1>**]=**<value1> **[**<key2>**]=**<value2> **[**<key3>**]=**<value3> ...**)**	definition of an associative array (dictionary – key-value pairs); each value in the array is associated with and accessed by a corresponding unique key; associative arrays must be explicitly declared before being defined
	`$ declare -A hostname_to_ip=(["server1"]="192.168.1.1" ["server2"]="192.168.1.2" ["server3"]="192.168.1.3")`
	(declares and defines an associative array "hostname_to_ip" and assigns it three different keys with their values)
	`$ hostname_to_ip["server4"]="192.168.1.4"` | `$ hostname_to_ip+=(["server4"]="192.168.1.4")`
	(appends an additional key-value pair to the associative array)

EXPANSION	**replaces specific strings or expressions with their corresponding values or results**
Brace expansion:	**process that generates combinations of strings or sequences based on the specified pattern**

{<string>[**,**<string>]**}**	braces; define a string (comma-separated strings) to be used in the pattern; the interval of consecutive numbers or letters is expressed by "**..**"; a prefix or suffix can be added to each string which becomes part of the expression
	`$ mkdir p{la,ri,oi}nt`
	(creates directories "plant", "print" and "point")
	`$ mv text{1,2,3,4,5}.txt text_0{1,2,3,4,5}.txt`
	(renames files "text*X*.txt" to "text_0*X*.txt")
	`# mv /etc/pam.d/sshd{,.broken}`
	(renames the file "/etc/pam.d/sshd" to "/etc/pam.d/sshd.broken")
	`$ touch .{a,b,c,d,e}`
	(creates hidden files of the specified names)
	`$ touch {1..100}`
	(creates one hundred files)
	`$ echo {A..Z} {a..z} {0..9}`
	(displays the interval of the specified characters)
	`$ echo {A..Z}{a..z}{0..9}`
	(displays three-figure combinations of characters in the specified order)

Tilde expansion:	**process that replaces a tilde character with the corresponding user's home directory path**
~	tilde; represents the logged-in user's home directory ($HOME), **~**<user> home directory of the specified user, **~+** working directory ($PWD), **~-** previous working directory ($OLDPWD) `$ cp ~/file.txt /tmp` (copies the "file.txt" file from the user's home directory to the "/tmp" directory)
Parameter expansion:	**process that replaces parameters with their values**

SPECIAL CHARACTERS	**have a special meaning, the interpreter is the shell**
$**<n> \| **${<n>**}**	the value of a positional parameter; if a positional parameter consisting of more than a single digit is expanded, it must be enclosed in braces `$ ./backup.sh /media/disk-1` (specifies the device for the backup – the script states e.g., "target_dir="$1"")
$**<parameter> \| **${<parameter>**}**	the value of a special parameter `$ echo $?` (prints the return code of the last executed command)

$**<variable> \| **${<variable>**}**	the value of a variable; a space is required after the variable, otherwise it must be enclosed in braces (except when followed by "**.**", "**$**", "**!**" or "**[]**")
	`$ echo $xy`
	(prints the value of the variable "xy" – "day")
	`$ echo it is a new week$xy` \| `$ echo "it is a new week$xy"`
	(prints "it is a new weekday")
	`$ echo "${xy}long period"`
	(prints "daylong period")
	`$ echo $SHELL`
	(prints the user's login shell)

${#<variable>**}**	the length of the variable's value (number of characters)

${<indexed_array>**[***n***]}**	the value of an indexed array's item; the indexed array must be enclosed in braces
	`$ echo "First colour: ${colours[0]}"`
	(prints "First colour: blue")
	`$ echo ${colours[@]}`
	(prints all items in the indexed array separated by a space)
	`$ for colour in "${colours[@]}"; do echo "Colour: $colour"; done`
	(prints each item in the indexed array on a separate line)

${#<indexed_array>**[@]}**	the length of the indexed array's values (number of items)

${<associative_array>**[**<key>**]}**	the value of an associative array's key; the associative array must be enclosed in braces
	`$ echo ${hostname_to_ip["server2"]}`
	(prints "192.168.1.2")
	`$ echo ${hostname_to_ip[@]}`
	(prints all values in the associative array separated by a space)
	`$ echo ${!hostname_to_ip[@]}`
	(prints all keys in the associative array separated by a space)
	`$ for server in "${!hostname_to_ip[@]}"; do echo "$server: ${hostname_to_ip[${server}]}"; done`
	(prints each key with its value in the associative array on a separate line)

${#<associative_array>**[@]}**	the length of the associative array's values (number of items)
Pathname expansion:	**process in which metacharacters are used to represent the names of existing files and directories that match the specified pattern**

*	asterisk; matches any number of any characters including spaces, except for a leading dot
	`# rm -Rf /*`
	(removes the contents of the directory)
	`$ ls *' '*`
	(lists files and directories containing a space in the name)

SPECIAL CHARACTERS	**have a special meaning, the interpreter is the shell**

?	question mark; matches any single character including a space, except for a leading dot
	`# find / -name "*.19??"`
	(finds files and directories with the extension ".19xx")
	`# find / -name "???*"`
	(finds files and directories whose names contain at least three characters)

[…**]**	square brackets; match any single character in the list; the list can consist of a specification of individual characters, an interval of characters (contains "**-**", e.g., A-Z, a-z or 0-9) or a class of characters ([:alnum:], [:alpha:], [:ascii:], [:blank:], [:cntrl:], [:digit:], [:graph:], [:lower:], [:print:], [:punct:], [:space:], [:upper:], [:word:], [:xdigit:]); "**!**" or "**^**" at the beginning of the list excludes the subsequent characters (groups of characters)
	`$ ls text_[ab].txt`
	(lists files "text_a.txt" and "text_b.txt")
	`$ find . -name "[a-zA-Z[:digit:]]"`
	(finds files and directories that contain only one lowercase or uppercase letter or digit in the name)
	`$ rm -f .[!.]*`
	(removes all hidden files in the working directory)
	`$ ls ~/.ssh/id_*[!.pub]`
	(lists all private ssh keys)

Arithmetic expansion:	**process that evaluates arithmetic expressions and replaces them with their results**

$((**<expression>**))** \| **$[<expression>**]**	dollar and double parentheses or a dollar and square brackets; the arithmetic expression inside is evaluated and replaced by its result
	`$ echo "2*5=$((2*5))"` \| `$ echo "2*5=$[2*5]"`
	(displays "2*5=10")
	`$ numbers=(10 20 30); echo "Sum of first two items = $((numbers[0] + numbers[1]))"`
	(displays "Sum of first two items = 30")
	`# uid=500; for usr in a b c; do useradd -u $uid $usr; uid=$(($uid+1)); done`
	(creates three accounts whose UID starts with 500 and increments by 1)

Command substitution:	**process that replaces a command with its output**
`` `<command>` `` | **$(**<command>**)**	backquotes (backticks) or a dollar and parentheses; the command inside is processed and replaced with its output `$ rpm -qf `which xhost`` | `$ rpm -qf $(which xhost)` (the output of the command "which xhost" is passed to the command "rpm -qf")
Process substitution:	**process that allows the output of a command to appear as input from a file or output to a file**

<(<list>**)** | **>(**<list>**)**	parentheses and a "less than" or "greater than" sign; execute the list of commands whose output is saved to a special temporary file (named pipe – FIFO) which is then passed as an argument to the current command that expects a file to read from or write to
	`$ diff <(hostname) /etc/hostname`
	(compares the contents of the command output (special file) with the contents of the specified file)
	`$ cat input.txt | tee >(grep a > a.out) >(grep b > b.out) >(grep c > c.out) > /dev/null`
	(filters data from the "input.txt" file based on the specified patterns and redirects them to corresponding separate files)
	`$ echo "$error_message" \| tee >(mail -r "$sender" -s "$subject" "$recipients")`
	(displays an error message and sends its contents by email)

REDIRECTION CHARACTERS	**redirect the standard input, output and error of a command**

SPECIAL CHARACTERS	**have a special meaning, the interpreter is the shell**
< | **0<**	standard input redirection from a file (STDIN / file descriptor 0) `$ mail tom@prompt.cz < list.txt` (writes an email to user "tom", the content of which is in the file "seznam.txt")

<< <delimiter>	standard input redirection from multi-line text (here-document)
	`#!/bin/bash`
	`cat <<EOF > /etc/myconfig.conf`
	`# My configuration file`
	`parameter1=value1`
	`parameter2=value2`
	`parameter3=value3`
	`EOF`
	(redirects multiple lines of text in a script up to the delimiter "EOF" to STDIN of the "cat" command)
	`$ cat <<END`
	`> I am $LOGNAME. <-'`
	`> END <-'`
	(interactive form)

<<< <string>	standard input redirection from a single line of text or a variable (here-string)
	`$ wc -w <<< "This is a line of text"`
	(counts the number of words in the string)
	`$ read -a members_array <<< "$members"`
	(reads the space-separated string stored in the variable "$members", splits it into individual words, and stores each word as an element in the array "members_array")

> | **1>**	standard output redirection to a file (STDOUT / file descriptor 1)
	`$ ls -la > dir_contents.txt`
	(redirects the command output to the specified file which is also created, if it exists, it overwrites its contents)
	`$ cat a | tee b > c`
	(redirects the command output to files "b" and "c")
	`$ wc -l < report > /tmp/lines`
	(writes the number of lines of the "report" file to "/tmp/lines")

2>	standard error redirection to a file (STDERR / file descriptor 2)
	`$ cat file1 file2 > file3 2> /dev/null`
	(writes the contents of files "file1" and "file2" to "file3" and discards the error output of the command)
	`$ make all 2> /dev/pts/3`
	(displays the standard error output of the command in the specified terminal)

2>&1	standard error redirection to standard output (it depends on the order of the STDOUT redirection)
	`$ ls a b > c 2>&1` | `$ ls a b &> c`
	(writes STDOUT and STDERR only to file "c")
	`$ ls file1 file2 2>&1 > file3`
	(displays STDERR on the screen and writes STDOUT to "file3")
	`$ make all 2>&1 | less`
	(especially useful when a process prints multiple error messages)

&>	standard output and standard error redirection to a file `# yum install -y libstdc++.x86_64 &> /dev/null` (discards all command output)
>>	standard output redirection to the end of a file `# echo 'ALL ALL=!SUDOSUDO' >> /etc/sudoers` (appends the command output to the end of the specified file, if the file does not exist, it is created)

SPECIAL CHARACTERS	**have a special meaning, the interpreter is the shell**
2>>	standard error redirection to the end of a file `# fsck /dev/sda1 2>> error` (appends the standard error output of the command to the end of the specified file, if the file does not exist, it is created)
QUOTING CHARACTERS	**suppress the meaning of special characters**
\	backslash (escape character); prevents the shell from interpreting the following single character `$ echo \$car` (prints "$car”)

'...'	single quotes; prevent the shell from interpreting any string inside the quotes; single quotes, however, cannot be enclosed within themselves
	`$ echo '$car'`
	(prints "$car”)
	`$ echo '\$car'`
	(prints "\$car”)
	`$ echo ''$car''`
	(prints "jeep”)

"..."	double quotes; similar to single quotes, but they allow the interpretation of substitution characters and "**$**" and "`" characters, "\" expands only when followed by "**$**", "**"**", "`", "\" and a new line; double quotes, however, cannot be enclosed within themselves
	`$ echo "$car"`
	(prints "jeep”)
	`$ echo "\$car"`
	(prints "$car”)
	`$ echo ""$car""`
	(prints "jeep”)

REGULAR EXPRESSIONS (extended)	**represent patterns that are used to match specific strings in the text, the interpreter is the program**
Special characters:	**specify the character of the expression**
.	dot; matches any single character including a space
[...**]**	square brackets; match exactly one character in the list, "**-**" specifies an interval of characters (e.g., A-Z, a-z or 0-9), "**^**” at the beginning of the list excludes the subsequent characters
[[:alpha:]] / **[a-zA-Z]**	alphabetic characters
[[:lower:]]	lowercase letters
[[:upper:]]	uppercase letters
[[:digit:]] / **[0-9]**	numeric characters
[[:alnum:]] / **[0-9a-zA-Z]**	alphanumeric characters
[[:punct:]]	punctuation characters
[[:print:]]	printable characters
[[:blank:]]	space or tabulator
(...**)**	parentheses; group characters
|	pipe (logical OR); separates searched patterns
\	backslash; suppresses the meaning of the following special character
Positional characters:	**specify the position of the expression**
^	caret; matches the beginning of a line
$	dollar sign; matches the end of a line
\<...\>	escaped angle brackets; match a pattern that is not surrounded by a letter, number or underscore

REGULAR EXPRESSIONS (extended)	**represent patterns that are used to match specific strings in the text, the interpreter is the program**
Quantifiers:	**specify the number of repetitions of the previous expression**
?	question mark; matches 0 or 1 times
*	asterisk; matches 0 or more times
+	plus; matches 1 or more times
{<n>**}**	number in braces; matches exactly *n* times
{<m>,<n>**}**	numbers in braces separated by a comma; matches at least *m* times, but not more than *n* times
{<m>,**}**	number and a comma in braces; matches at least *m* times
{,<n>**}**	comma and a number in braces; matches at most *n* times
Regular expression:	**matches:**
a.	"a" + any single character
a+b	"ab", "aab", "aaab" ...
a\+b	"a+b"
s?care	"scare" or "care"
auto(mobile)?	"auto" or "automobile"
micro(phone|scope)	"microphone" or "microscope"
^(Subject|From):	a line starting with the string "Subject:" or "From:"
ha{1,3}	"ha" or "haha" or "hahaha"
<T[DH]>	"<TD>" or "<TH>"
\<a.*a\>	a string starting and ending with "a"
\[[a-zA-Z]\]	any letter enclosed in "[]"
[0-9]+	at least one digit
[0-9]|[1-9][0-9]	a range of numbers "0"–"99"
[0-9]{2}	a range of numbers "00"–"99"
(19|20)[0-9]{2}	a range of numbers "1900"–"2099"
[0-9a-fA-F]|[1-9a-fA-F][0-9a-fA-F]+	hexadecimal numbers
^$	an empty line
[.^az\-]	a dot, caret, "a", "z", backslash or dash
[^ ,.]+	a string that does not contain a space, comma, or dot
.+0$	a line ending with "0" preceded by at least 1 character
^P.*(0[1-9])$	a line starting with "P" and ending with "01"–"09"
(\(?(\+|00)[0-9]{1,4}\)?)?([-]?[0-9]{2,4}){2,4}([[:blank:]]|$)+	any phone number
[a-zA-Z0-9_.-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}	any email address
https?://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}(/[^]*)?	any web address