Brian Will Terminal & Shell Playlist

Part 1:

- Terminals (terminal emulator) are dumb devices to handle/show ASCII text input and to interact with the shell programm
- Process communicate with the terminal through a character device file(CDF) (write to buffer)
- Processes inherit after forking two standard fd: stdin (0) reading from CDF and stdout(1) writing do same (CDF) —> 2 needed because of redirection
-

Part2:

- There are different shell programs. shell programs takes user input from terminal, interprets as command and execute commands. Shell is an interactive programming language.
- BASH (Bourne again shell —> extension of sh of Stephen Bourne)
- PROMPT: *brian@ubuntu:~$* —> user@OS:current_path($ —> end of promt)
- \n —> end of command —> shell execute the input
- Process command: ls -la bin
  - ls: programm name
  - -la argument 1 (flag/option)
  - bin argument 2
- where to find the programs to execute:
  - ls: look all the paths in the environment variable $PATH
  - /bin/ls: absolute path in root dir

    - bin/ls: relative from the current working dir (the pwd: print working directory), only when at least one / is in the path. ./foo oder ../foo possible as short form
- personal programs can be added to the PATH to execute them without path
- sbin: contains programs for superusers
- shell executes command in the following way:
    - forking itself and copying, address to the arguments (located on heap) and the argc on the stack its variables and file descriptors
    - forking parent waits for the child
    - forked child executes the command
- There are characters with special meaning - METACHARACTER - in bash syntax:
    - # ‘ “ \ $ `* ~? < > ( ) ! | & ; space newline
- Quoting: Represent the literal meaning of the char (ignoring its special meaning)
    - \ :escape
    - ‘ ‘ : quotes every character inside
    - “ “: quotes every character inside except of $ ‘ \ ! * @ - they keep their special meaning

Part 3:

- Terminal is the default stdin and stdout of a created (forked) process. The process reads from the the terminal and writes to it
- Redirection: Change the location to read from / to write to
    - < filepath: Redirection of stdin for reading
    - > filepath: Redirection of stdout for writing
    - Redirection can be placed anywhere in the command
- Piping allow interprocess communication. The output of one process gets the input of another process when they are connected via a pipe and the fd are set up accordingly with redirection:
    - command A | command B: command A reads from stdin, writes to the pipe, command B reads from the pipe, writes to stdout

    - pipeline: one ore more cmds connected via a pipe —> parallel: command A | command B
    - command list: one ore more pipelines terminated by a ; & or \n —> sequential command A; command B
- exit code of a program: 0=OK, non-0=error (exit syscall):
    - example: pipeline A && pipeline B, only exec pipeline if pipeline A exits 0
    - example pipeline A || pipeline B, B gets not executed if A exits already 0
- built-in commands:
    - process cmds get executed by forking the shell, built-ins are directly implemented in the shell code itself, redirection still possible but implemented in another way. Every built in gets a duplicate of stdin/stdout fd, the duplicates are redirected
    - *help*: get help on built ins
    - cd - sets the shells current working directory: has to be built in, because a forked process would not change the original shells pwd but the one of the forked process
    - cd with no args —> home directory
    - echo: Print args to stdout —> useful with argument variable expansion
- variable expansion
    - set variable: name=value
    - view all variables on zsh: typeset
    - echo $name —> looks up the the value of the variable and replaces it
    - foo=4; echo ${foo}d —> prints 4d
    - if variable doesn't exists it resolves to an empty string “”
    - Quoting:
        - echo '$foo' —> prints $foo
        - echo “$foo” —> prints value of foo
    - create environment variable with export:

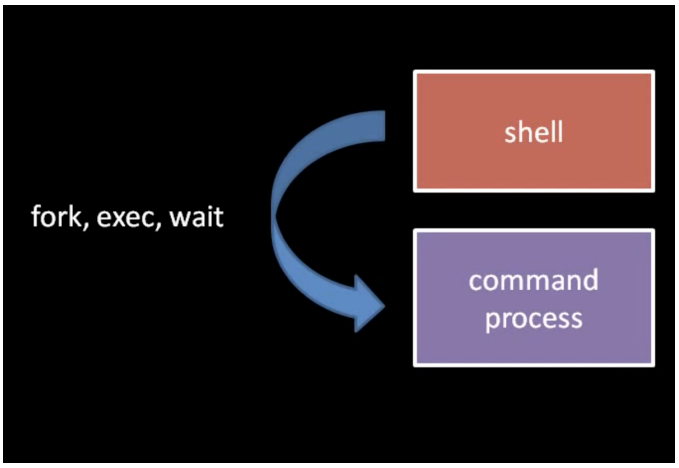
    - foo=8; export foo (environ var is created); foo=Hallo (gets overwritten)
    - shell vars are only present in the shell they were created, environment variable ar visible to forked processes

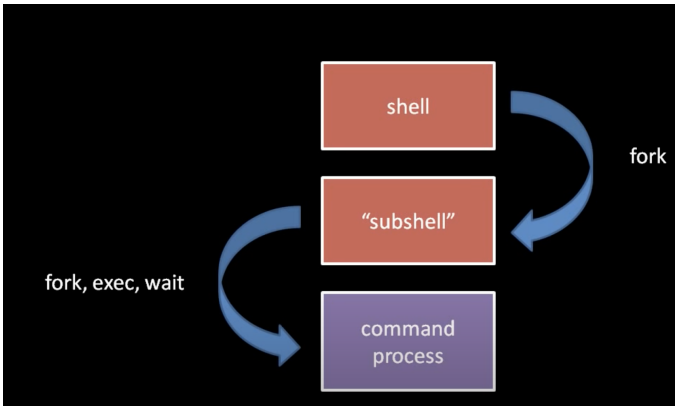
Part 4:

- functions in shell - not necessary atm
- brace expansion
    - foo{apple, banana}bar —> fooapplebar foobananabar
    - {hi, you}hi —> hihi youhi
- tilde expansion
    - tilde = shorthand for home dir
- command substitusion
    - $(command A $(command B)): The result of command B (printed to stdout) replaces the command itself and gets an arg to command A, the replaced command is run in a subshell
- arethmetic substitution
    - $((expression)): echo $((3 + 5)) —> 8
- filename expansion
    - *: Asterisk wildcard —> match any characters
    - ?: match any single character; foo?bar —> fooDbar is valid, fuuDbar is not
- Order of expansion and substitution:
    - brace exp, tilde exp, variable exp, arithmetic exp, command sub, filename exp

Part 5:

Shell vs subshell execution





- to execute in a sub shell put command list ind parens : (command-list): *(cd /; ls -la) —> no effects on the current shell*
- to execute in the current shell, put in curly braces: { command list }, curly brace is built in command, therefore is a space after the opening curly brace is needed. The brace is the name of the built in.
- &-terminated pipeline runs in background subshell, shell doesn't wait
    - foo & bar ; fizz ; buzz &
        - exec foo and does not wait and executes bar immediately. Waits the for finishing and starts fizz. Waits for fizz and starts buzz and does not wait.