B4B33PSY: Počítačové systémy

Lekce 13. Bonusová přednáška
cmake Vim tmux

Petr Štěpán

stepan@fel.cvut.cz

6. prosince 2025

Motivace – osnova

- CMake
  - Jak nám pomůže cmake při přeladu projektu
- Vim
  - Velmi krátký úvodu do vimu
  - Efektivní editoru pro terminál
- tmux
  - Multiplexovaný terminál
  - Vhodný způsob spuštění komplikovaných systémů přes ssh

Co je CMake

CMake

- **CMake** je nástroj pro správu sestavování projektu
  - CMake neprovádí samotnou kompilaci
  - CMake **generuje** tzv. build soubory (např. Makefile, Ninja, Visual Studio projekty)
- CMake je **platformně nezávislý**
  - pracuje na všech známých systémech - Linux, Windows, macOS
- Vstupem pro CMake jsou konfigurační soubory
  - konfigurační soubory mají standardně jméno `CMakeLists.txt`

Proč CMake vznikl?

- Makefile jsou:
  - obtížně udržovatelné
  - závislé na platformě
  - špatně škálovatelné pro větší projekty
- CMake řeší:
  - přenositelnost
  - modularitu
  - automatickou správu závislostí

Základní princip - CMake

1. Vývojář napíše soubor **CMakeLists.txt**
2. CMake analyzuje:
   - zdrojové soubory
   - závislosti
   - cíle (programy - executables, knihovny)
3. CMake **vygeneruje** build systém (např. Makefile)
4. Vlastní kompilaci provede nástroj:
   - `make`
   - `ninja`
   - MSBuild

Rozdíl mezi CMake a Make

CMake	Make
Generátor build systému	Vlastní build nástroj
Platformně nezávislý	Platformně závislý
Vyšší úroveň abstrakce	Nízká úroveň
Snadnější údržba	Obtížná údržba
Vhodný pro velké projekty	Spíše menší projekty

- **Make** je nástroj, který:
  - **čte** Makefile
  - provádí zadané příkazy, např. kompiluje zdrojové soubory
  - řeší závislosti mezi soubory
- Makefile:
  - obsahuje konkrétní příkazy pro kompilátor
  - je silně závislý na platformě

Vztah CMake a Make

- CMake **nenahrazuje** Make, ale:
  - **generuje** Makefile
- Typický způsob použití je:
  1. `mkdir build`
  2. `cd build`
  3. `cmake ..`
  4. `make`
- CMake lze použít i bez Make, např. s nástrojem Ninja, nebo Visual Studio
- Make lze použít bez CMake, pokud si napíšete Makefile sami

CMake

Hlavní výhody CMake:

- Jednotná konfigurace pro více platforem
- Snadná správa závislostí
- Modulární struktura projektů
- Podpora velkých projektů
- Integrace s IDE (CLion, Visual Studio, VS Code)
- Je velmi rozšířený pro C/C++ projekty

Nevýhody CMake:

- Vlastní jazyk (nutnost se ho naučit)
- Složitější syntaxe pro začátečníky
- Chyby se často projeví až při generování výsledku - buildu

Kdy použít CMake

- Multiplatformní projekty
- Střední a velké C/C++ projekty
- Projekty s více knihovnami
- Projekty s externími závislostmi

Základní struktura CMakeLists.txt

Uživatel musí nadefinovat vše, co je potřebné k sestavení projektu:

- Definice projektu - název projektu a verze
- Nastavení jazyka a požadované verze překladače
- Definice zdrojových souborů
- Definice cílových souborů
- Použití knihoven třetích stran
  - kde jsou hlavičkové soubory knihovny
  - kde jsou binární soubory knihovny
  - případně, jaká minimální verze knihovny je potřeba

CMake minimální projekt

CMakeLists.txt minimální projekt:

```
cmake_minimum_required(VERSION 3.16)

project(MyProject)

add_executable(myexample simple.cpp)
```

cmake_minimum_required - důležité uvést aktuální verzi CMake se kterou pracujete

- CMake se umí nastavit zpětně na zadanou verzi a tím být plně kompatibilní

project - definice vytvářného balíčku

- CMake nemusí být pro jediný program, může to být balíček programů nebo knihoven

CMake minimální projekt

```
project(MyProject
  VERSION
    1.0
  DESCRIPTION
    "Very nice project"
  LANGUAGES
    CXX
)
```

add_executable - nedefinování programu jako výsledku projektu

- jako parametry se zadává název programu a veškeré zdrojové soubory pro tento projekt (i hlavičkové soubory)
- alternatině je možné vytvářet knihovnu klíčovým heslem **add_library**

Nastavení překladače v CMake

Jak nastavit konkrétní překladač, např. clang nebo gcc?

- CMake použije překladač, který je nastaven v BASH prostředí, ze kterého voláte `cmake`
- Pokud tedy chcete nastavit překladač na clang, proveďte následující příkazy před zavoláním cmake:

```
export CC=/usr/bin/clang
export CXX=/usr/bin/clang++
```

V CMakeLists.txt lze detailně nastavit i parametry překladače:

```
target_compile_options(myexample PRIVATE -Wall -Wextra -
Wunreachable-code -Wpedantic)
```

Nastavení překladače v CMake

Lze použít i následující upřesnění:

- **target_link_libraries** knihovny, které se mají dynamicky připojit k uvedenému cíli
- **target_include_directories** další adresáře s hlavičkovými soubory
- **target_compile_features** vlastnosti překladače, které chcete použít, např. cxx_std_11
- **target_compile_definitions** definice, které se mají použít pro překlad, parametry -Ddefinice
- **target_compile_options** ostatní nastavení překladače

Nastavení překladače v CMake

Pokud chcete použít překladač určitého standardu, můžete využít nastavení proměnných:

```
set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)
```

CMake projekt s OpenCV

Zadání projektu s využitím cizí knihovny:

- Jednoduchá C++ aplikace
- Používá knihovnu **OpenCV**
- Načte obrázek a zobrazí ho v okně

Struktura projektu:

```
opencv_project/
├── CMakeLists.txt
└── main.cpp
```

Soubor `main.cpp`

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>

int main() {
    cv::Mat image = cv::imread("image.jpg");

    if (image.empty()) {
        std::cerr << "Nelze načíst obrázek" << std::endl;
        return 1;
    }

    cv::imshow("Obrazek", image);
    cv::waitKey(0);
    return 0;
}
```

Soubor `CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.16)

project(OpenCVExample VERSION 1.0 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Najdi OpenCV
find_package(OpenCV REQUIRED)

# Přidej cestu k hlavičkovým souborům OpenCV knihovny
include_directories( ${OpenCV_INCLUDE_DIRS} )

# Vytvoř spustitelný soubor
add_executable(opencv_example main.cpp)

# Přilinkuj OpenCV knihovny
target_link_libraries(opencv_example PRIVATE ${OpenCV_LIBS})
```

Vysvětlení CMake konfigurace

- `find_package(OpenCV REQUIRED)`
  - Najde nainstalovanou OpenCV knihovnu
  - Lze přidat minimální verzi knihovny
- `${OpenCV_INCLUDE_DIRS}`
  - Automaticky nastavené cesty k adresářům s hlavičkovými soubory
- `${OpenCV_LIBS}`
  - Automaticky nastaví cesty potřebné k sestavení programu s knihovnami
- CMake řeší:
  - include cesty
  - linker nastavení

Build a spuštění projektu

```
mkdir build
cd build
cmake ..
cmake --build .
./opencv_example
```

Výhody použití CMake s OpenCV

- Žádné ruční nastavování cest pro správné nalezení hlavičkových souborů knihovny OpenCV
- Funguje na mnoho platformách - Linux / Windows / macOS
- Snadná integrace s IDE
- Jednoduché rozšíření o další zdrojové soubory

Chyby při vytváření `CMakeLists.txt`

Co se pokazí nejčastěji:

- **CMake nenajde OpenCV**
  - Chyba: `Could NOT find OpenCV`
  - Řešení:
    - OpenCV není nainstalované, nebo není správně nainstalované
      - Nastavit proměnnou `OpenCV_DIR` (hlavně Windows)
    - použít program `pkg-config` pro kontrolu správnosti instalace knihovny
      - `pkg-config --cflags --libs opencv`
    - pokud si knihovnu překládáte sami ze zdojových kódů je např. pro OpenCV od verze 4 potřeba zkontrolovat nastavení `-DOPENCV_GENERATE_PKGCONFIG=YES`

Chyby při vytváření `CMakeLists.txt`

- **Chyby při sestavování - linkování**
  - Chyba: undefined reference / unresolved external symbol
  - Řešení:
    - použít `target_link_libraries` pro všechny knihovny, které je nutno k programu připojit
    - nepsat ručně linker flagy
- **Špatná verze C++ standardu**
  - OpenCV vyžaduje novější C++
  - Řešení:
    - `set(CMAKE_CXX_STANDARD 17)`

Chyby při vytváření `CMakeLists.txt`

- **Spuštění programu bez potřebných DLL (Windows)**
  - Program se nespustí
  - Řešení:
    - chyba při instalaci knihovny OpenCV
      - Windows - přidat OpenCV DLL do PATH nebo zkopírovat DLL k .exe
      - Unix - zkontrolovat nástroj `ldconfig`, který je zodpovědný za to, že OS nalezne dynamickou knihovnu
        - pro modifikaci dynamických knhoven je potřeba oprávnění `root`
- **Chybná cesta k souborům (např. obrázek)**
  - `imread` vrací prázdný obrázek
  - Řešení:
    - spouštět program z build adresáře
    - používat absolutní cesty

Shrnutí CMake

- CMake = **moderní standard pro build systémy**
- Zjednodušuje vývoj a údržbu
- Odděluje konfiguraci od samotné kompilace
- Výrazně lepší a jednodušší než ruční Makefile
  - pokud nefunguje, pak si můžete sestavit vlastní Makefile
- Jednoduše lze přidat do projektu i další knihovny, např. na trénování neuronových sítí, na matematické výpočty.
- Jednoduše lze přidat vytvoření testů
  - V projektu lze vytvořit více programů, tedy i testů, které testují funkčnost modulů výsledného programu

Alternativy CMake

Přestože je CMake velmi rozšířen pro prakticky všechny C/C++ projekty, existují další podobné systémy, které se snaží být uživatelsky příjemější:

- Meson
  - uživatelsky příjemnější konfigurace projektu
  - rozšířen hlavně na Linuxu
    - méně multiplatformní
- Bazel
  - pro velké projekty
  - umí kombinovat různé jazyky - C/C++, Python, Java a další
  - pracuje na Linuxu, Windows
  - v některých směrech je složitější než CMake

Kdy zvolit který nástroj

- **CMake**:
  - maximální kompatibilita
  - velké a dlouhodobé projekty
- **Meson** nebo **Bazel**:
  - nové projekty
  - důraz na jednoduchost
- **Ninja**:
  - rychlost buildu
  - alternativa k Make
  - pouze jako výstup CMake, Meson nebo Basel
  - komplikovanější než Makefile

Úvod

Vim

- **Vim** (Vi IMproved) je výkonný textový editor dostupný na většině platforem
- Navazuje na editor **vi** (Unixová tradice)
- Ovládání primárně z klávesnice ⟶ vysoká efektivita pokud se ho naučíte
- Široce používaný jako editor přes ssh, vzdálená správa počítačů, vývoji softwaru a v akademickém prostředí

Proč používat Vim

- Rychlá práce bez myši
- Nízké hardwarové nároky
- Dostupný na vzdálených počítačích přes SSH
- Extrémně přizpůsobitelný
  - Mnoho vlastností lze nakonfigurovat
- Silná komunita a množství pluginů

Základní koncept - Módy

Vim je **modální editor** – chování kláves závisí na aktuálním módu.

Hlavní módy:

- **Normal mode** – pohyb, příkazy (výchozí mód)
- **Insert mode** – psaní textu
- **Visual mode** – výběr textu
- **Command-line mode** – příkazy začínající `:`

Přepínání:

- `Esc` ⟶ Normal mode
- `i`, `a`, `o` ⟶ Insert mode
- `v`, `V`, `Ctrl+v` ⟶ Visual mode
- `:` ⟶ Command-line mode

Spuštění a ukončení Vim

- Spuštění Vimu se současným otevření souboru:

  ```
  vim soubor.txt
  ```

  - Pokud soubor neexistuje, tak se vytvoří prázdný soubor
  - Příkazem `:e nazev.txt` otevřete soubor `nazev.txt` místo aktuálního souboru
  - Příkazem `:e!` znovu načtete otevřený soubor
- Častěji se Vim ukončí a otevře s jiným souborem:
  - `:q` – zavřít
  - `:q!` – zavřít bez uložení
  - `:w` – uložit
  - `:wq` / `ZZ` – uložit a zavřít

Pohyb v textu (Normal mode)

Základní klávesy:

- `h` ←, `j` ↓, `k` ↑, `l` →
- lze používat i šipky, ale někdy si blízký a vzdálený terminál nerozumí
  - vzdálený terminál špatně interpretuje zakódované infomrace o rozšiřujících klávesách jako jsou funkční klávesy F1-12, nebo i šipky či PgUp a PgDn
  - v tom případě je nutné použít alternativu v podobě písmen

Rychlejší navigace:

- `w` / `b` – další / předchozí slovo
- `0` / `$` – začátek / konec řádku
- `gg` / `G` – začátek / konec souboru
- `Ctrl+f` / `Ctrl+b` – stránkování

Editace textu

Vkládání textu - přepnutí do insert módu:

- `i` – před kurzor
- `a` – za kurzor
- `o` – nový řádek pod

Mazání - v Normal módu:

- `x` – znak
- `dd` – řádek
- `dw` – slovo

Kopírování a vkládání:

- `yy` – kopírovat řádek
- `p` – vložit za kurzor
- `P` – vložit před kurzor

Undo, redo a vyhledávání

- `u` – zpět (undo)
- `Ctrl+r` – znovu (redo)

Vyhledávání:

- `/text` – hledat
- `n` / `N` – další / předchozí výskyt

Nahrazování:

```
:%s/staré/nové/g
```

Práce s více soubory

- Otevření více souborů:

```
vim a.txt b.txt
```

- Přepínání:
  - `:n` / `:prev`

Okna a panely:

- `:split`, `:vsplit`
- `Ctrl+w` + směr

Vim pluginy

Pro efektivnější vyžívání Vimu existuje mnoho modulů s předprogramovanými funkcemi - **Plugin**

- **startify** - plugin, který Vám umožní pokračovat v poslední práci, případně obsahuje rychle dostupné nejčastější soubory, které editujete
- **coc** - Conquer Of Completion - autocomplete, dokončování kódu podobně jako znáte ve VS Code
- **ale** - Asynchronous Lint Engine - kontrola kódu, vyhledávání a dokončování názvů symbolů v programu
- **fzf** - Fuzzy Finder - nalezení souboru podle částečně zadaného jména, také rychlé prohledávání více souborů
- **vim-multiple-cursors** - psaní a více místech souboru současně, vhodné na přejmenování proměnných, paralelní úpravu textu
- mnoho pluginů pro barevné reprezentace Vimu a grafické úpravy výstupu

Výhody a nevýhody Vim

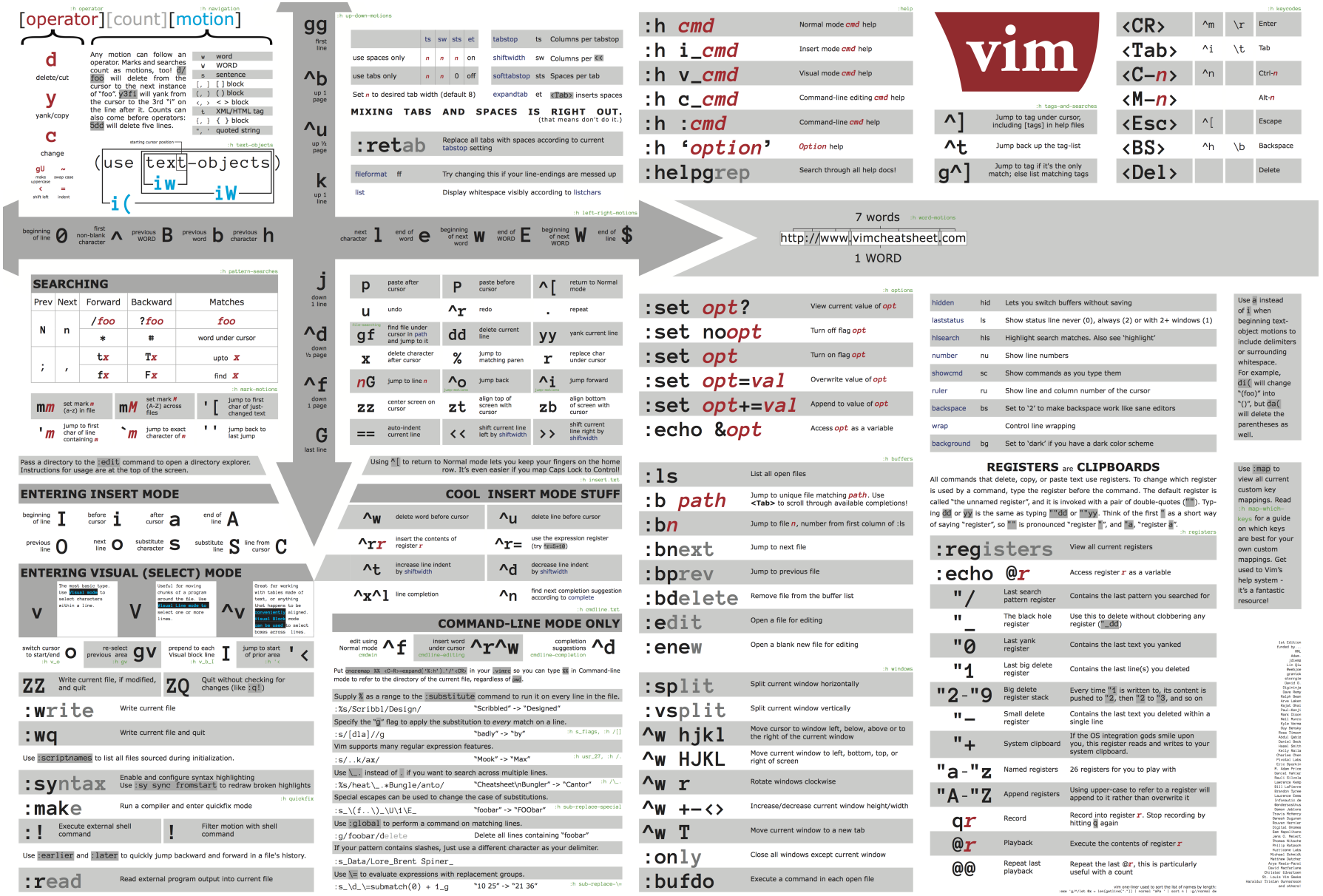
Výhody:

- Extrémní efektivita
- Přenositelnost
- Plná kontrola nad editorem

Nevýhody:

- Strmá křivka učení
- Neintuitivní pro začátečníky

Vim Cheat Sheet



Učení Vimu

Jak se naučit Vim?

- každodení používání Vimu
  - alespoň jednou týdně použijte nový příkaz
- `vimtutor` (interaktivní kurz)
- Vim games:
  - hravý způsob výuky
  - Vim Hero, Vim Genius, Vim Racer, …
  - Vim Adventures - https://vim-adventures.com
- https://www.vim.org
- Dokumentace ve Vimu `:help`

Vim - závěr

- Vim je silný nástroj pro editaci textu i programování
- Vyžaduje čas na naučení, ale výrazně zvyšuje produktivitu
- Díky konfiguraci a pluginům se přizpůsobí každému uživateli

Tmux - Úvod

Tmux

- **tmux** (terminal multiplexer) je nástroj pro práci s více terminálovými relacemi v jednom okně
- Umožňuje rozdělit terminál na panely, přepínat mezi sezeními a pracovat vzdáleně
- Velmi populární mezi vývojáři, administrátory a DevOps
- Časté použití v robotice
  - jedno ssh připojení spustí předem definovaný počet virtuálních terminálů
  - každý terminál je pro spuštění jedné části řídicích programů
  - jednoduché přepínání mezi virtuálními terminály a kontrola jejich funkčnosti

Tmux úvod

Proč používat tmux?

- Práce s více terminály současně při jediném přihlášení
- Přetrvání relace po odpojení (např. SSH)
  - v robotice systém nezkolabuje při ztrátě komunikace přes WiFi
- Efektivní práce bez myši
- Zvýšení produktivity

Základní pojmy

- **Session (sezení)** – samostatné pracovní prostředí
- **Window (okno)** – karta v rámci sezení
- **Pane (panel)** – rozdělení okna na části
- **Prefix** – klávesová zkratka pro ovládání tmux (výchozí `Ctrl+b`)

Instalace tmux

- Linux (Debian/Ubuntu): `sudo apt install tmux`
- macOS (Homebrew): `brew install tmux`
- Ověření instalace: `tmux -V`

Základní ovládání

- Spuštění tmux: `tmux`
- Odpojení od sezení – detach: `Ctrl+b d`
- Ukončení panelu: `exit`
- Nápověda kláves: `Ctrl+b ?`

Tmux používání

Práce se sezeními

- Vytvoření nového sezení: `tmux new -s jmeno`
- Seznam sezení: `tmux ls`
- Připojení k sezení: `tmux attach -t jmeno`
- Ukončení sezení: `tmux kill-session -t jmeno`

Okna (Windows)

- Nové okno: `Ctrl+b c`
- Přepínání oken: `Ctrl+b n` / `Ctrl+b p`
- Seznam oken: `Ctrl+b w`
- Přejmenování okna: `Ctrl+b ,`

Panely (Panes)

- Vertikální rozdělení: `Ctrl+b %`
- Horizontální rozdělení: `Ctrl+b "`
- Přepínání panelů: `Ctrl+b šipky`
- Zavření panelu: `Ctrl+b x`

Kopírovací režim

- Aktivace: `Ctrl+b [`
- Pohyb pomocí klávesnice
- Kopírování textu
- Užitečné pro práci bez myši

Konfigurace tmux

Konfigurační soubor: `~/.tmux.conf`

- Možnost změny prefixu
- Vlastní klávesové zkratky
- Barevná schémata a status bar

tmuxinator program, který čte soubor typu yaml a podle obsahu provede spuštění a nastavení tmux session:

- textový soubor
- definice oken a panelů
  - spuštění příkazů v oknech a panelech

Konfigurace tmux session

- Příklad `tmuxinator.yaml` konfiguračního souboru z MRS systému pro drony:

```
name: simulation
socket_name: mrs
attach: false
tmux_options: -f /etc/ctu-mrs/tmux.conf
pre_window:
  - export UAV_NAME=uav1
  - export RUN_TYPE=simulation

startup_window: status
windows:
  - core:
      layout: tiled
      panes:
        - waitForHwApi; ros2 launch mrs_uav_core core.launch.py
  - status:
      layout: tiled
      panes:
        - ros2 run mrs_uav_status status.sh
  - edge_detector:
      layout: tiled
```

Konfigurace tmux session

```
    panes:
      - ros2 launch example_edge_detector edge_detector.launch.py
- takeoff:
    layout: tiled
    panes:
      - waitForCore; sleep 5; ./takeoff.sh
- rviz:
    layout: tiled
    panes:
      - waitForCore; ros2 run rviz2 rviz2 -d ./rviz.rviz
```

Výhody a nevýhody

Výhody

- Lehký (HW nenáročný) a rychlý
- Funguje přes SSH
- Vysoce přizpůsobitelný

Nevýhody

- Podobně jako Vim - pomalá učící křivka
- Ovládání založené na klávesových zkratkách

Praktické využití

- Robotické aplikace
  - Vhodné pro ROS - Robot Operating System
    - Podle scénaře spuštění řídicích programů a uživatelských rozhraní, které by jinak bylo neúměrně náročné
- Správa serverů
- Dlouhotrvající procesy

PSY shrnutí

Děkuji za pozornost a doufám, že jste se něco nového naučili

- Omlouvám se za pozdní zadání úkolů

Rád bych od Vás získal zpětnou vazbu ohledně:

- přínosu přednášek
  - která přednáška Vám přišla nejzajímavější
  - která byla nejhorší, co Vám v předmětu chybělo
- cvičení
  - na co byste chtěli více času
  - čemu se věnovat méně
  - čemu se věnovat jiným způsobem
  - co byste vypustili, co byste přidali
- domácí úkoly
  - který se Vám líbil nejvíc
  - který byste vynechali
  - odpovídá bodové hodnocení - navrhněte změnu