TEX v jednoduchém UNIXovém prostředí

Petr Olšák

Abstrakt. Při ladění TEXového dokumentu potřebujeme mnohokrát opakovaně pouštět TEX, podívat se, jak dopadl výsledek v prohlížeči DVI nebo PDF souboru, mrknout na výpis TEXu na terminálu, podívat se případně do logu a celou činnost opakovat. V tomto článku je ukázáno, jak tuto práci dělá autor článku. Proces „editor-TEX-kuk“ je zde podporován jednoduchými UNIXovými nástroji: bashovým skriptem `texloop`, který si autor pro tyto účely vytvořil, dále terminálem Xterm a jednoduchým editorem, který umí navázat na klávesovou zkratku spuštění příkazu v systému. Čtenář se zde může inspirovat a přizpůsobit tyto nástroje svým vlastním potřebám. V článku je popsána funkce skriptu `texloop`, dále je neformálně rozveden dlouholetý vývoj autorova vztahu k textovým editorům a konečně je zde uvedena konfigurace terminálu Xterm, aby vyhovoval českému prostředí jak v kódování ISO-8859-2, tak v kódování UTF-8. Pro kódování UTF-8 si v závěru článku vygenerujeme TEXový formát `csplain`.

Skript `texloop`

Bashový skript `texloop` se chová jako démon ve smyslu, že čeká na signály. Ovšem protokol o své činnosti vypisuje na terminál. Můžeme ho tedy pustit v nějakém terminálu a v jiném terminálu pustit textový editor. Pokud v editoru uložíme zdrojový soubor a vyšleme signál (to můžeme udělat stiskem jediné klávesy, stačí si na to připravit klávesovou zkratku v editoru), démon `texloop` se probudí, spustí požadovanou TEXovou úlohu, a přitom na „svůj“ terminál vypisuje hlášení TEXu. Pokud úloha dopadla bez chyby, pošle démon signál prohlížeči DVI nebo PDF a ten automaticky obnoví zobrazení. Takže na jedinou klapku v editoru TEXujeme a hned vidíme výsledné změny v prohlížeči.

Démon `texloop` spustíme příkazem

```
texloop tex-command main-file
```

kde „`main-file`“ je název TEXovaného souboru bez přípony, například tedy

```
texloop csplain mujtext
```

V první fázi činnosti je `texloop` v interaktivním módu. Spustí TEX podle zadaného příkazu se zadaným vstupním souborem. Pokud TEX narazí na chybu, zastaví se a čeká na terminálu na reakci. Komunikaci můžeme samozřejmě ukončit pomocí všem TEXistům známé klávesy `X`. Ovšem to uděláme s tím rizikem, že nemusí vzniknout soubor pro prohlížení. Jak se v takovém případě `texloop` zachová, popíšu za chvíli.

Souborem pro prohlížení je soubor `main-file.pdf` v případě, že `tex-command` začíná na písmena `pdf`. Například tedy `pdftex`, `pdfcsplain`, `pdflatex`. Jinak je souborem pro prohlížení `main-file.dvi`.

Pokud existuje po prvním interaktivním běhu TEXu soubor pro prohlížení, `texloop` spustí odpovídající prohlížeč tohoto souboru (`xpdf` nebo `xdvi`) a přejde do „daemon“ módu, kdy čeká na signály. Když soubor pro prohlížení neexistuje, žádný prohlížeč se zatím nespustí a `texloop` i v tomto případě přejde do „daemon“ módu. Pokud `texloop` po některém z příštích běhů TEXu zjistí, že soubor pro prohlížení dodatečně vznikl, neváhá v daném okamžiku spustit odpovídající prohlížeč.

V „daemon“ módu `texloop` čeká na signály. Rozlišuje pouze dva: `SIGUSR1` pro spuštění TEXu a `SIGTERM` pro ukončení činnosti. Obdrží-li signál `SIGUSR1`, pak zkontroluje časy poslední

modifikace souborů `main-file.tex` a `main-file.log`. TEXová úloha se spustí právě tehdy, když je zdroják novější než log. Výjimku z tohoto pravidla uvedu za chvíli.

TEXová úloha v „daemon“ módu běží v neinteraktivním režimu. Při výskytu první chyby TEX ukončí svoji činnost. Možnost pokračování činnosti TEXu a výpisu všech chyb jsem zavrhnul, protože dnes jsou počítače natolik rychlé, že nikoho nezdržuje po opravení první chyby spustit TEX znovu. Osobě tento režim práce preferuji.

Jestliže běh TEXu skončil úspěšně, `texloop` požádá prostřednictvím signálu prohlížeč TEXového výstupu, aby znovunačetl PDF resp. DVI a obnovil zobrazení. V každém případě po ukončení běhu TEXu přechází `texloop` znovu do stavu čekání na signály.

Proces `texloop` končí svou činnost buď po obdržení signálu `SIGTERM` (můžeme zmáčknout `Ctrl-C` v terminálu, kde `texloop` běží), nebo ukončením prohlížeče PDF nebo DVI prohlížeče, který byl procesem `texloop` spuštěn.

Někdy se může stát, že editujeme jiný soubor, než „hlavní“ soubor `main-file.tex`. Představme si, že je v `main-file.tex` příkaz `\input` a ten načítá nějakou kapitolu dokumentu. Textovým editorem jsme zrovna zalezlí do této kapitoly, měníme ji a chceme vidět výsledek zpracování celého dokumentu. Uložení kapitoly a vyslání signálu pro `texloop` nepomůže, protože `texloop` zjistí, že je log novější než zdroják a na TEXování se vydlábne. Můžeme ovšem potlačit vykonání testu, kterým `texloop` kontroluje stáří zdrojáku a logu. Stačí přidat k parametrům pro `texloop` písmeno `A` (jako „run TEX Always“). Například tedy

```
texloop csplain kniha A
```

Abychom mohli daemonu `texloop` vyslat signál, musíme vědět, jaký má PID. Program `texloop` v okamžiku přechodu z interaktivního do „daemon“ módu uloží svůj PID do souboru `~/.tlpid`, odkud si jej může editor přečíst. Když ovšem spustíme další proces `texloop` (vedle už běžícího), pak nový proces přepíše obsah tohoto souboru svou hodnotou. Editor si tedy musí „zavčas“ informaci převzít. Druhá možnost je, že editor vyšle signál všem procesům `texloop` daného uživatele a konkrétním PIDem procesu se nezabývá. Pak stačí, když použije příkaz `killall -USR1 texloop`. Nevýhoda tohoto přístupu je, že možná některé procesy budou TEXovat zbytečně, ale není to obvyklé. Procesy bez příznaku `A` (Always) totiž budou TEXovat jen v případě, že je zdroják novější než log.

Skriptík `texloop` je docela jednoduchý. Má 80 řádků, najdete ho na [1] a můžete si ho přiohnout k obrazu svému.

Textové editory, aneb konečně končím s Emacsem

V této kapitole rozvedu neformálním způsobem své zkušenosti s textovými editory. Pro lidi, kteří očekávají spíše nezaujaté technické informace a zejména pro fanoušky Emacsu není tato kapitola určena. Pro uživatele MS Windows také není tato kapitola určena. Tito uživatelé jistě už poznali, že jim je k ničemu kompletně celý článek.

S interaktivními textovými editory jsem se poprvé setkal v DOSu a poté v UNIXu. Jiné operační systémy umožňující interakci s uživatelem jsem neměl tu čest poznat.

V DOSu jsem po dlouhém vývoji zakotvil u Qeditu, který jsem si přizpůsobil obrazu svému. Jeho výchozí chování bylo zřejmě inspirováno editorem WordStar. Qedit měl přehledný konfigurační soubor s možností jednoduché tvorby maker a klávesových zkratek. Ovšem to bylo zhruba před dvaceti lety. Při přechodu na UNIX jsem se naučil aspoň základy editoru `vi`. Tento editor měl totiž tu vlastnost, že se dal najít na skoro každém UNIXu, i kdyby tam už nebylo vůbec nic jiného. Setkal jsem se výjimečně i s UNIXy natolik ořezanými, že tam nebyl ani `vi`. Editovat konfigurační soubory systému v řádkovém editoru `ed` pak byla skutečná lahůdka. Tehdy nás ale nic nemohlo odradit od našeho nadšení z UNIXu.

Až do roku 2011 jsem používal dva editory: `vi` a Emacs. První jmenovaný jsem používal jako „odlehčený editor“ pro editaci systémových souborů. „Odlehčený editor“ musí být snadno dostupný v každém UNIXu a musí umět spolupracovat s běžně používanými terminály. Nesmí se opírat o přítomnost Xserveru, protože ten často není při konfiguraci systému přítomen. Umím jen úplné základy editoru `vi` a podobně jako plno jiných lidí mě psychicky nesmírně vyčerpává přepínat mezi dvěma módy tohoto editoru. Ale co, v době počátku UNIXů nic jiného nebylo.

Druhý jmenovaný editor (Emacs) je kapitola sama pro sebe. Před dvaceti lety měla jeho uživatelská dokumentace 700 stránek, dnes jsou k Emacsu dokumenty dva, dohromady mají 1600 stránek. Tím se mi nikdy nepodařilo prokousat. Komu se to podařilo, nechť se přihlásí a pak hodí kamenem. Domnívám se, že editor má sloužit a ne být středem pozornosti a nutit uživatele mu věnovat neúměrnou pozornost. Emacs má velmi neobvyklý způsob ovládání, který s mým oblíbeným Qeditem z DOSu má pramálo společného. Také používá naprosto neobvyklou terminologii, takže najít něco konkrétního v dokumentaci metodou `grep` je téměř nemožné. Nevíte totiž, co máte hledat. Navíc je to komplikováno tím, že Emacs má dokumentaci uloženu ve svém svérázném formátu, ke kterému nabízí přístup svým svérázným způsobem (programem `info`). Uvedu namátkou jen tři terminologické svéráznosti. Klávesovou zkratku `Ctrl-A` Emacs značí „`C-a`“. Klávesu `Alt` nazývá „Meta“ a značí „`M`“. Symbol „`M-A B`“ v dokumentaci znamená „zmáčkni `Alt` a `Shift`, zmáčkni `A`, pusť `Alt` a stále drž `Shift`, zmáčkni `B`“. Kdo nepustí klávesu `Alt` ve správném okamžiku, má smůlu, Emacs vykoná něco jiného. Nikomu není příjemné, když se mu Emacs takto směje do ksichtu.

Protože Emacs byl v té době jediný editor, který byl použitelný na rozsáhlejší věci (včetně TEXování rovnou z editoru), investoval jsem do něj asi měsíc svého života a pokusil se vytvořit alespoň pár vlastních klávesových zkratek na TEXování a pro aspoň trochu normální ovládání. I poté jsem některé složitější úkoly (např. posuny sloupcových bloků) v Emacsu udělat neuměl. Další čas strávený nad dokumentací, než bych dokázal potřebnou informaci vyhledat, se mi hrubě nevyplatil. Pokorně jsem tedy pustil ve vedlejším okénku DOS emulátor, v něm nahodil Qedit a úkol vyřešil tam.

Asi před deseti lety přišla další nepříjemnost v podobě přechodu z verze Emacsu 19 na verzi vyšší. Mnou pracně vytvořená makra v nové verzi samozřejmě nefungovala. Dokumentace se stala ještě obludnější a nesrozumitelnější. Nějakou dobu jsem kromě implicitní verze Emacsu v UNIXové distribuci udržoval vedle i starou verzi 19, na které jsem provozoval veškerou svou práci. Bohužel, jak roky běžely, byla verze 19 stále obtížněji kompilovatelná pro nové distribuce. Zpočátku jsem se dokázal povrtat v Céčkových zdrojích a kompilaci nějak rozdýchat, ale s příchodem dalších verzí kompilátoru a zejména novějších knihoven to šlo stále hůř. Takže jsem se rozhodl znovu investovat pár týdnů svého času, ponořit se do zrůdné dokumentace a vytvořit pokud možno tytéž klávesové zkratky, jako jsem používal pro Emacs 19. Od jednoho emacsového guru jsem se dozvěděl další nepříjemnost: nová verze už nebude umět jedním klapnutím opravit špatně napsaný kus řádku, který člověk hledící do klávesnice (místo na monitor) napsal omylem v nespr8vn2 p5epnut0 kl8vesnici. Cituji názor tohoto emacsového guru: naučte se psát všemi deseti a dívat se při psaní na monitor.

Všechny UNIXové editory, se kterými jsem se setkal, trpí dalším neduhem: předpokládají, že člověk sedí u vyorané klávesnice z doby kamenné, která nemá funkční klávesy na pohyb kurzoru. Takže nejstručnější klávesové zkratky řeší posun kurzoru. Například `Ctrl-F` je posun o jedno políčko doprava. Ha, jak skvělé, ale jak k ničemu! Začátečnický tutoriál k Emacsu je rovněž plný takových totálně zbytečných informací. I tohle komplikuje vztahy mezi editorem a jeho potenciálními uživateli.

Měl jsem možnost mluvit s jedním příznivcem Emacsu. Zeptal jsem se ho, jak se dá přepnout wordwrap mód z implicitního emacsího chování, kdy je potřeba psát dlouhé slovo poslepu třeba za roh a teprve mezera za slovem toto slovo ulomí na další řádek. Očekával bych normální chování, kdy se slovo ulomí v okamžiku, kdy kurzor dosáhne nastavené hranice. Dále jsem se chtěl nechat poučit, jak efektivně v dokumentaci zjistit některé věci. Třeba jak vypnout case-insensitive mód při povelu `isearch`. Nebo jak zkrotit syntaktické zvýrazňování. Pro TEXové zdrojáky přišel Emacs s novinkou, že texty za podtržítkem se zobrazují zmenšeně (jako indexy). To působí velmi rušivě. Jak to opravit a nepřijít o zvýrazňování? Ani na jednu otázku jsem nedostal uspokojivou odpověď.

Emacs je z mého pohledu obluda, která se vymyká **základnímu zákonu UNIXu**, který praví: *UNIXové prostředí sestává z malých elementárních programů, které dělají jen jeden dílčí úkol, kvůli kterému byly navrženy, a ten dělají dobře. Tyto programy vzájemně spolupracují.* Mám na mysli třeba `grep`, `awk`, `bash`, `bc`, `tar`, `sed`, `rsync`, TEX a další. Editor Emacs byl prvním významným narušitelem tohoto základního zákona. Pak přišli další, o OO se nebudu raději zmiňovat.

Z výše uvedeného vyplývá, že jsem při editování souborů dvacet let trpěl. Dokonce jsem pomýšlel na to, že si napíšu editor vlastní. Ale letos přišlo vysvobození. Začalo to tím, že jsem si instaloval nově systém do notebooku. Jako první potřebuji mít odlehčený editor na editování konfiguračních souborů systému. V Gentoo Linuxu je pro tento účel přednastaven editor `nano`, který ale člověka pravidelně vytáčí dvěma blbými otázkami, než uloží pozměněný soubor a ukončí činnost. Přeci jenom `vi` editor se svým `Shift-ZZ` vykoná tuto práci bez ptaní a hned. Editor `vi` v čisté podobě už dávno neexistuje, používal jsem tedy už dlouho jeho `vim` implementaci. A v nové verzi mě `vim` velmi rozhněval. Implicitně má nastaveno na kurzorové klávesy vlevo/vpravo chození po slovech. V tomto režimu jsem vůbec nebyl schopen editovat. Řekl jsem si, že pokud do deseti minut v dokumentaci či pomocí Google nenajdu, jak se tato nová skvělost odstraňuje, letí `vim` z mého počítače pryč. A taky že letěl.

Čím ale `vi` nahradit? Před deseti lety jsem viděl v provozu editor Joe, takže vím, že to je další možný odlehčený editor. Během pár vteřin po jeho nainstalování jsem věděl, že bez blbého ptaní uložím soubor a ukončím činnost editoru pomocí `Ctrl-KX`. Během pár minut jsem shledal, že editor za těch deset let výrazně pokročil kupředu. Během hodiny už začínám tušit, že toto je editor, který mi nahradí nejen odlehčený editor, ale také Emacs. Má všechny funkce, které pro své editování potřebuji. Vše jsem se dozvěděl v kratičké manové stránce, online nápovědě a v přehledně vyvedeném vzorovém konfiguračním souboru `/etc/joe/joerc`. Jako třešničku na dortu uvedu skutečnost, že Joe má 25 krát menší binárku než Emacs. A dělá přesně to, k čemu byl určen, a dělá to dobře. V souladu se **základním zákonem UNIXu**. Spočítejte si, kolik knihoven jednotlivé binárky načítají dynamicky. Emacs jich potřebuje ke svému životu 44, zatímco editoru Joe stačí tři základní knihovny (`libc`, `libm` a `libutil`). Při kompilaci se dá nastavit ještě závislost na `ncurses`, ale není to nutné.

Níže uvádím seznam vlastností, které mě u Joe editoru potěšily. Mnohé z nich jsem nikdy nedokázal v Emacsu nastavit. Netvrdím, že by to v Emacsu nešlo, vždyť „Emacs umí všechno“. Spíš je to kvůli mé neschopnosti se orientovat v nepřehledné emacsí dokumentaci.

- Při rychlém hledání i při najdi-nahraď vnímá rozdíly mezi velkými a malými písmeny. Dá se to podle potřeby vypnout.
- Kurzor se nesnaží lepit na konce řádků, takže při posunování dolu/nahoru neskáče při své pozici vpravo jako čamrda.
- Při rolování dolů/nahoru obraz s sebou neškube.
- Wordwrap se chová normálně.
- Snadno přepnu do hexa módu.

- V UTF8 terminálu snadno editor přepnu do stavu, kdy edituje ISO-2 soubory a naopak v ISO-2 terminálu zase mohu editovat UTF8 kódované soubory.
- Neřeší, co řešit nemusí: fonty, přepínání klávesnice. To je starost terminálu.
- Snadno přeskočím během editování na místo v souboru, které jsem nedávno opustil a pak se zase snadno vrátím tam, kde jsem byl předtím.
- Editor si pamatuje polohu kurzoru před ukončením. Když vlezu do souboru po nějaké době znovu, startuje v místě, kde jsem soubor opustil. Taky si trvale pamatuje použité příkazy pro překlad a další věci. Historií těchto příkazů mohu procházet šipkami nahoru/dolů.
- Neptá se blbě na každou pitomost.
- Únik ze všeho pomocí `Ctrl-C` je přirozený. Výchozí klávesové zkratky se podobají zkratkám v mém starém dobrém Qeditu (tj. jsou odvozeny z WordStaru).
- Má jednoduchou konfiguraci nových zkratek a maker. Během chvíle jsem si například nastavil propojení editoru s filtrem `vlna`, což byl úkol, na který jsem z hlediska jeho obtížnosti v případě Emacsu po celých dvacet let rezignoval a vlnkoval soubory „ručně“ ve vedlejším terminálu.
- Zobrazí `^M` na koncích řádků. Setkávám se totiž občas se soubory majícími pomršené konce řádku (z DOSu nebo z MS Windows). Pokud se mě editor snaží odstínit od tohoto problému, je to špatně, neboť pak vznikají záhadné chyby například v shellových skriptech. Na druhé straně je editor schopen se přizpůsobit (když chci) a editovat v DOSovém módu konců řádků.
- Konečně zase umím snadno označit sloupcové bloky (i myší) a posunovat s jejich obsahem.
- Syntaktické zvýrazňování je příjemné a má jednoduché a přehledné konfigurační soubory. Během chvíle jsem si vylepšil například syntaktické zvýrazňování TEXových zdrojáků k obrazu svému.
- Editor se naučí slova použitá v načteném souboru a umožní jejich rychlé doplňování. Stačí napsat začátek slova a zmáčknout příslušnou klávesu.

Nebudu zde uvádět všechny podstatné vlastnosti editoru. Stručný seznam jeho možností najdete na [2]. Myslím si, že editor nabízí vše, co běžný uživatel (jako jsem já) potřebuje pro normální práci.

Některé nevýhody editoru mohou plynout z faktu, že je zcela závislý na prostředí terminálu, ve kterém je spuštěn. Na vstupu tedy snímá ty informace, které mu tam pošle terminál. Například Xterm vysílá při zmáčknutí klávesy Backspace stejný kód jako při zmáčknutí klávesy Delete a to je stejný kód jako při `Ctrl-H`. Mezi těmito třemi klapkami pak editor nemůže rozlišit a reaguje na všechny jako Backspace. Chtěl bych vidět toho člověka, který vyvíjel termcap pro Xterm, když tam navrhnul Delete=Backspace. Asi před tímto rozhodnutím se mu srazila hlava s železničním pražcem. Problém jsem si opravil úpravou konfigurace Xtermu, ale od této chvíle nemám Xterm splňující oficiální parametry.

Klávesové zkratky `Alt-něco` se nedají použít, neboť je terminál převede na jednobytovou informaci typicky shodnou se zmáčknutím nějaké klapky s akcentovaným znakem. Editor se vůbec nedozví, že byla zmáčknuta klávesa `Alt`. Znamená to, že můžeme pracovat pouze s klávesovými zkratkami typu `Ctrl-něco`. Aspoň se to lépe pamatuje. V případě zkratek `Ctrl-něco` máme k dispozici jen 32 možností, při kterých terminál generuje kódy 0 až 31 takto: `Ctrl-@` je nula, `Ctrl-A` je 1, `Ctrl-B` je 2, atd., `Ctrl-^` je 30 a `Ctrl-_` je 31. Kdo zatím nevidí souvislost, podívá se za domácí úkol na ASCII tabulku. Přitom ještě platí, že `Escape=Ctrl-[`. Máme ovšem štěstí, že klávesy `F1`, `F2` atd. a některé další vysílají z terminálu k aplikaci poněkud delší sekvenci znaků (uvozenou kódem Escape), podle které aplikace rozpozná, co bylo zmáčknuto. Při zmáčknutí `Ctrl-F1` je vygenerována jiná escape sekvence než při `F1`. Takže

tyto zkratky typu `Ctrl-Fněco` dokáže editor rozpoznat a dokonce nespadají do omezujících 32 možností zmíněných dříve.

Různé terminály vysílají bohužel při mačkání kláves F1 a spol. různé escape sekvence. Někdy se tyto sekvence dokonce překrývají v různém významu. Například RxVT a Xterm mají vzájemně prohozeny Home a End. Editor Joe bohužel neumí pracovat s konfigurací escape sekvencí závislou na použitém terminálu. Já používám výhradně Xterm a Linuxovou konzoli a pro tyto dva typy terminálů jsem si upravil svou konfiguraci Joe editoru.

Výše popsané omezení plyne z toho, že chceme použít odlehčený editor uvnitř terminálu, tj. třeba i na vzdáleném počítači s mizerným síťovým připojením. Zkuste si Emacs pustit uvězněný uvnitř terminálu a shledáte, že se za jistých okolností chová stejně. Žádnou klávesu `Meta` alias `Alt` nerozliší.

Další možnou nevýhodou Joe editoru je jeho až příliš jednoduchá konfigurace maker. Jsou k dispozici jen jednoduché podmínky, není možná žádná práce s proměnnými, jen sled za sebou jdoucích elementárních pokynů pro editor. Abych tedy mohl v Joe příjemně TEXovat, vytvořil jsem si (mimo jiné) bashový skriptík `texloop`.

Někdo může za nevýhodu Joe editoru považovat také skutečnost, že se jeho autor asi před dvěma lety vytratil z Internetu. Když jsem mu nedávno posílal návrh na jednoduchou změnu ve formě záplaty, pošta se mi vrátila jako nedoručitelná kvůli přeplněné emailové schránce. Osobně by mi vůbec nevadilo, kdyby se vývoj editoru zastavil. Vychytávky, které si v konfiguraci editoru udělám, pak budou fungovat navždycky.

Zdrojové kódy Joe editoru jsou napsány ve srozumitelném jazyce C, takže jsem si sám dokázal opravit jednu drobnou chybku a doprogramovat reakci na prostřední tlačítko myši. Chtěl bych zdůraznit, že toto je výhoda *jednoduchého* a *otevřeného* software. Uživatel si jej může přiohnout k obrazu svému. Musí ale být splněny současně obě dvě vlastnosti. Například zdrojáky Otevřené kanceláře jsou sice otevřeny, ale vykostit z nich algoritmy na konverzi kancelářských formátů do PDF a udělat z nich dávkové programy pro příkazovou řádku se ještě nikomu nepodařilo.

Tím, že jsem opustil Emacs a `vim` a přešel konečně na normální editor, jsem si výrazně ulehčil život. Vymizela schizofrenie, zda v dané chvíli chci TEXovat nebo editovat konfigurační soubory vzdáleného serveru. Dělám to od této chvíle stejným editorem s jednotným způsobem ovládání. Autor Joe editoru Joseph H. Allen se přesně strefil do mých představ, jak má tento typ software vypadat. Možná by měl mít tento článek jiný název: The Joy Of Joe.

Uživatelská konfigurace editoru

Uživatelská konfigurace Joe editoru je uložena (přirozeně) v souboru `~/.joerc`. Tam si můžeme definovat výchozí hodnoty přepínačů, jednoduchá makra a vazby na klávesové zkratky. Kromě toho je možné založit adresář `~/.joe` a do něj vložit další soubory konfigurace: konfiguraci závislou na různých typech souborů, konfiguraci syntaktického zvýrazňování pro různé jazyky apod. Při tvorbě uživatelské konfigurace je možné se inspirovat výchozí konfigurací `/etc/joe/joerc` nebo použít konfiguraci, kterou jsem si vytvořil pro svou potřebu a kterou jsem zveřejnil na [1]. V této kapitole stručně popíšu vlastnosti této uživatelské konfigurace.

Klávesové zkratky jsou trojího typu: „jednoklapkoidní“ (např. `Ctrl-C`), „dvouklapkoidní“ (např. `Ctrl-KX`) a zkratky uvozené klávesou Escape (např. `Esc L` – spuštění korekce překlepů pomocí programu `aspell`). Všechny jednoklapkoidní zkratky, které v originální konfiguraci řeší pohyb kurzoru, jsem považoval za neobsazené, protože pohyb kurzoru realizuji pomocí šipek a pomocí kláves Home, End, PgUp, PgDn. Některé z takto uvolněných zkratek jsem obsadil tím, co mi připadalo přirozenější. Například: `Ctrl-S` – rychlé vyhledávání vpřed, `Ctrl-R` – rychlé vyhledávání vzad, `Ctrl-Z` – undo, `Ctrl-U` – vyvrhnutí naposledy smazaného

textu na místo kurzoru, `Ctrl-P` – skok na předchozí (vzdálenou) pozici kurzoru, `Ctrl-J` – zrušení označeného bloku. Přesný přehled všech těchto změn je v krátkém dvojstránkovém shrnutí na [1].

Editor umožňuje (kromě maker v souboru `.joerc`) nahrávat další uživatelská makra s čísly 0 až 9 za provozu a pak je přehrávat pomocí dvouklapkoidní zkratky `Ctrl-Kčíslo`. Někdy se ale hodí přehrát makro jednoklapkoidním hmatem, abychom je mohli přehrávat se stejnou rychlostí, na kterou máme nastaven opakovací kmitočet klávesnice. Proto jsem zkratku `Ctrl-K\` navěsil na přehrání makra 0.

Dvouklapkoidní zkratky typu `Ctrl-Kněco` jsem nechal v původním významu. Jsou odvozeny z WordStaru a já jsem se s nimi dávno seznámil při práci s Qeditem. Taktéž zkratky uvozené klávesou Escape jsem nechal v původním významu. Je tedy možné je dohledat v online nápovědě editoru, která je dostupná pomocí `Ctrl-KH`. Na další stránku nápovědy se přechází pomocí `Esc tečka`. To sice není příliš praktické, ale zase tak často v nápovědě listovat nepotřebujeme.

Nově jsem přidal dvouklapkoidní zkratky typu `Ctrl-Xněco`. Zkratka `Ctrl-XC` uloží vyznačený blok do souboru `~/.joetmp` a následně zkratka `Ctrl-XV` si obsah tohoto souboru vyzvedne. Tím je možné zprostředkovat přenos bloku mezi dvěma různými instancemi Joe editoru spuštěnými na stejném počítači.

Přenos bloku je samozřejmě možné provést i pomocí clipboardu X windows systému. Ovšem pozor. V konfiguraci mám zapnutý přepínač `mouse`. Takže myš označuje bloky přímo pro Joe editor, nikoli pro clipboard. Joe editor neumí s clipboardem přímo pracovat (to by musel být slinkovaný s knihovnami X window systému, což po něm opravdu nechceme). Blok Joe editoru lze nicméně zkopírovat do clipboardu pomocí zkratky `Ctrl-N` (k tomu se využije externí program `xclip`). Nebo můžeme při označování bloku přidržet klávesu `Shift`. V takovém případě se do clipboardu dostane text označený a viděný v Xtermu bez toho, aby se o tomto označení Joe editor vůbec dozvěděl. Xterm mu totiž tuto aktivitu zatají. Pro vyzvrácení clipboardu do místa kurzoru je možné použít `Shift`-prostřední tlačítko myši. Samotné prostřední tlačítko myši nedělá nic (při kompilaci z originálních zdrojáků). Nebo kopíruje označený blok Joe editoru (při použití mé záplaty před kompilací).

Zkratka `Ctrl-X~` prožene vyznačený blok nebo celý soubor programem `vlna`, tj. doplní vlnky za neslabičné předložky. Dále zkratka `Ctrl-XH` opraví v naposledy vlo6en0m textu probl0my zp;soben0 Mistrem Janem Husem. Tyto problémy mají lidé, kteří se chvíli nedívají na monitor, píšou a posléze zjistí, že měli blbě přepnutou klávesnici. Mezi tyto lidi patřím i já. Pojem „naposledy vložený text“ zahrnuje text vložený na jediném řádku po předchozím pohybu kurzoru nebo po zahájení editování souboru. Pokud ovšem vyznačíme nějaký blok a pak použijeme `Ctrl-XH`, korekce proběhne ve vyznačeném bloku.

Editor je nastaven tak, že se nepokouší interpretovat kódování načteného textu (mohl by, ale tuto funkci jsem v konfiguraci vypnul). Uvnitř ISO-2 terminálu editor předpokládá kódování ISO-8859-2 a uvnitř UTF-8 terminálu předpokládá UTF-8 kódování. Je-li načten soubor v nesprávném kódování, poznáme to podle paznaků. V takovém případě je možno použít zkratku `Ctrl-XE`, která v ISO-2 terminálu zkonvertuje buffer z UTF-8 do ISO-8859-2 a naopak v UTF-8 terminálu zkonvertuje buffer z ISO-8859-2 do UTF-8.

Funkce vyvolané klávesovými zkratkami `Ctrl-N` (kopie clipboardu), `Ctrl-XH` (Hus) a `Ctrl-XE` (korekce kódování) volají skripty, které jsem si pro tyto účely vytvořil a uložil do adresáře `./joe` (a zveřejnil na [1]). Tyto skripty pracují různým způsobem v závislosti na nastaveném parametru `charmap` v locales. Tedy v závislosti na použitém terminálu (ISO-2 nebo UTF-8). Je potřeba vědět, že pokud přejdeme v editoru do „křížového“ kódování, tj. editujeme

např. ISO-8859-2 kódovaný soubor v UTF-8 terminálu, pak uvedené skripty nebudou pracovat správně.

Tak jako dříve v Emacsu, jsem soustředil i v Joe editoru TEXování na klávesu `F7`. Pomocí `Ctrl-F7` zahajuji proces `texloop` v nově založeném okně editoru. Editor si pamatuje posledně vložený příkaz (i příkazy starší), takže typicky nemusím vkládat celý příkaz `texloop` znovu. Poté mohu okno s procesem `texloop` skrýt (`F6`, `Ctrl-KI`) a mohu TEXovat pomocí `F7`. Skoro okamžitě po zmáčknutí této klávesy se mi obnoví zobrazení ve vedlejším prohlížeči PDF nebo DVI výstupu. Přitom (na rozdíl od Emacsu) okno s výpisem TEXu neobtěžuje, když nechci, ale mohu se tam podívat (`Ctrl-KI`).

Strasti a slasti s terminálem Xterm

Moje pracovní prostředí na počítači sestává z plochy Xserveru a na ní se vesměs rozprostírá spíše více než méně Xtermů. Klávesnice je implicitně anglická. Opravdu si nedovedu představit práci s příkazovým řádkem v české klávesnici. Jedna pracovní plocha Xserveru typicky nestačí, ovšem manager `fvwm` mi nabízí přepínat mezi devíti různými plochami. K tomu slouží klávesy `Ctrl-šipka`. To je důvod, proč jsem tyto sekvence nekonfiguroval jako klávesové zkratky pro editor. Občas na některém z devíti ploch je zapnuté i něco jiného než Xterm (například prohlížedlo www stránek, prohlížedlo PDF souborů atd.). Počet grafických programů ale nikdy nepřekročí množství Xtermů na pracovních plochách.

Uživatele počítačů se z mého pohledu dělí do dvou skupin, podle toho, jak přistoupí například k úloze „chci prohlédnout dokument v souboru `cosi.pdf` v adresáři `kdesi`“. První skupina spustí někde z nabídky window manageru program `xpdf` nebo něco podobného, pak tam v nabídce najde funkci „otevřít soubor“ a pak dlouze a podle abecedy hledá v periskopickém výpisu adresářů a souborů (s typicky zoufale malým průzorem) ten správný adresář a v něm ten správný soubor, pak klikne, a úloha je vyřešena. Druhá skupina vleze do Xtermu, v něm do příkazového řádku napíše „`cd kdesi Enter`“ (přitom velmi často využívá doplňování dlouhých názvů tabelátorem) a konečně napíše „`xpdf cosi.pdf`“. Nebo to urychlí a rovnou píše „`xpdf kdesi/cosi.pdf`“. Přiznám se, že já patřím do té druhé skupiny uživatelů. Proto potřebuji tolik Xtermů. Nutí-li mi někdo způsob práce první skupiny, jsem velmi nešťastný zejména z periskopických hledáčků obsahujících výpisy souborů. Pravda, existuje ještě třetí skupina uživatelů, která typicky neví, co to je PDF, neví, co to je soubor, co to je adresář a používá jen metodu „klikání na obrázky“. Tato skupina využívá též toho, že window manager má jasno, který soubor otevřít kterým programem. Tato skupina uživatelů je mi ovšem už natolik myšlenkově vzdálená, že ji do svých úvah dále nebudu zahrnovat.

Pokud potřebuji během své práce výjimečně přepnout do české klávesnice, pak pouze v jednom zvoleném Xtermu, nikoli pro všechny X-ové aplikace globálně. Klávesnici tedy přepínám Xtermem. Používám k tomu konfiguraci Xtermu vytvořenou už velmi dávno Liborem Škarvadou, Petrem Mejzlíkem a Janem Kasprzakem. Pozor, vývojáři Xtermu v nejnovějších verzích cosi pokonili a přestala fungovat mrtvá klávesa. Je tedy potřeba mít verzi Xtermu nejvýše 256.

Přepnutí klávesnice tam i zpět se provede klávesou `Pause`. Při zapnuté České klávesnici kurzor bliká, při vypnuté je kurzor v klidu. Tím lze poznat stav klávesnice v každém Xtermu. Klávesou `Ctrl-Pause` změním implicitní černé pozadí Xtermu na bílé. Rád koukám do černé barvy, ale když je vedle rozsvícený bílý prohlížeč, je kontrast černá-bílá příliš unavující pro oči a v takovém případě přepnu Xterm na bílou.

Popsaná konfigurace Xtermu je pro kódování ISO-8859-2 v souboru `XTerm` a pro kódování UTF-8 v souboru `UXTerm`. Definici české klávesnice jsem v souboru `UXTerm` musel kompletně přepsat. Tyto soubory si můžete stáhnout z [1] a umístit do svého systému. Kam, to je závislé na

systému. Já je mám typicky uloženy v `/usr/share/X11/app-defaults/`. Místo plných názvů fontů jsou v souboru `XTerm` uvedeny aliasy, které je potřeba nastavit v souboru `fonts.alias`. Jak, to také závisí na systému.

Na stránce [1] je též podrobný popis instalace Xtermu. Bohužel mám špatné zkušenosti s přenosem češtiny v clipboardu, pokud se použije v ISO-2 locales Xterm, který je kompilován i pro UTF-8. Mám proto v systému binárky dvě. Jednu, která je kompilovaná jen pro jednobytové kódování a druhá, která je kompilovaná pro UTF-8. Pak přenos v clipboardu probíhá spolehlivě i mezi různě kódovanými aplikacemi. Interní obsah clipboardu je vždy UTF-8.

V popisu instalace je zmíněna i možnost mít v systému (třeba na ploše vedle sebe) oba Xtermy (ISO-2 i UTF-8). Implicitně mám nastaveno v systému `LC_CTYPE=cs_CZ.ISO8859-2` a pro UTF-8 kódovaný Xterm přepínám do `LC_CTYPE=cs_CZ.UTF-8`. Ostatní proměnné z locales (včetně proměnné `LANG`) nejsou nastaveny, protože samozřejmě nechci, aby na mě příkazový řádek štěkal česky. To snad nechce žádný soudně uvažující uživatel.

UTF-8 kódovaný csplain

Formát `csplain` používám implicitně v kódování ISO-8859-2. Výhody přechodu na kódování UTF-8 nejsou v případě `csplainu` vůbec zřetelné. Znaky mimo znakovou sadu ISO-8859-2 stejně musíme individuálně ošetřit makry v TEXu. Takže mě nic nenutí explicitně přecházet na jiné kódování.

Někdy se ovšem stane, že zákazník dodává soubory v UTF-8 a my je chceme bez potřeby neustálého překódování zpracovávat v csplainu. Pak je možné samozřejmě tyto soubory zpracovávat v UTF-8 kódovaném terminálu. Místo příkazu `csplain` pak použiji příkaz `ucsplain` (respektive `pdfucsplain`). To je formát, který překódovává vstup z UTF-8 do ISO-8859-2 na úrovni input procesoru TEXu a využívá k tomu encTEX. Výstup na terminál, do logu a do `\write` souborů se vrací zase v kódování UTF-8. Formát vygenerujeme takto:

```
pdftex -ini -enc \\let\\enc=u \\input csplain.ini
mv csplain.fmt ucsplain.fmt
mv csplain.log ucsplain.log
pdftex -jobname pdfcsplain -ini -enc \\let\\enc=u \\input csplain.ini
mv pdfcsplain.fmt pdfucsplain.fmt
mv pdfcsplain.log pdfucsplain.log
su
cp ucsplain.* pdfucsplain.* /var/lib/texmf/web2c/pdftex/
texhash
cd /usr/bin
ln -s pdftex ucsplain
ln -s pdftex pdfucsplain
```

Nyní jsou všem uživatelům v systému přístupné příkazy `ucsplain` a `pdfucsplain`, kterými mohou zpracovávat zdrojové soubory kódované v UTF-8.

Odkazy

[1] `http://petr.olsak.net/texloop.html`
[2] `http://joe-editor.sourceforge.net/`