

Jak si usnadnit opakované spouštění T_EXu

Petr Olšák

Při ladění T_EXového dokumentu potřebujeme mnohokrát opakovaně pouštět T_EX, podívat se, jak dopadl výsledek v prohlížeči DVI nebo PDF souboru, mrknout na výpis T_EXu na terminálu, podívat se případně do logu a celou činnost opakovat. V tomto článku ukážu, jak tuto práci dělám na UNIXu. Vytvořil jsem si pro to bashový skript, jehož funkce je v článku podrobně vysvětlena.

Na kolotoč „editor-T_EX-kuk“ jsem zhruba dvacet let používal Emacs, ovšem s tímto editorem jsem nikdy nebyl spokojen. O svých trápeních s Emacsem a konečném přechodu na jiný editor píše v příloze k tomuto článku. Nový editor není vybaven mocným jazykem lisp (který ovšem skoro nikdo neumí) a má daleko jednodušší možnosti tvorby maker. Proto jsem si vytvořil bashový skript, který mi nakonec nabídl daleko větší komfort, než jsem měl dříve při práci s Emacsem.

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 pomocí jediné klávesy, stačí si na to připravit klávesovou zkratku v editoru), démon `texloop` se probudí, spustí požadovanou T_EXovou úlohu, a přitom na „svůj“ terminál vypisuje hlášení T_EXu. Pokud úloha dopadla bez chyby, pošle démon signál prohlížeči DVI nebo PDF a ten automaticky obnoví zobrazení. Takže na jedinou klapku v editoru T_EXujeme a hned vidíme výsledné změny v prohlížeči.

Démon `texloop` spustíme příkazem

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

například tedy

```
texloop csplain mujtext
```

V první fázi činnosti je `texloop` v interaktivním módu. Spustí T_EX podle zadaného příkazu se zadaným vstupním souborem. Pokud T_EX narazí na chybu, zastaví se a čeká na terminálu na reakci. Komunikaci můžeme samozřejmě ukončit pomocí všem T_EXistům 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` je jedna z možností: `pdftex`, `pdfcsplain`, `pdflatex`, `pdfcslatex`. Jinak je souborem pro prohlížení `main-file.dvi`.

Pokud existuje po prvním interaktivním běhu T_EXu 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 T_EXu 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 T_EXu a `SIGTERM` pro ukončení činnosti. Obdrží-li signál `SIGUSR1`, pak zkontroluje časy poslední modifikace souborů `main-file.tex` a `main-file.log`. T_EXová úloha se spustí právě tehdy, když je zdroják novější než log. Výjimku z tohoto pravidla uvedu za chvíli.

T_EXová úloha v „daemon“ módu běží v neinteraktivním režimu. Při výskytu první chyby T_EX ukončí svoji činnost. Možnost pokračování činnosti T_EXu a výpisu všech chyb jsem zavrhnul,

protože dnes jsou počítače natolik rychlé, že nikoho nezdržuje po opravění první chyby spustit T_EX znovu. Osobně tento režim práce preferuji.

Jestliže běh T_EXu skončil úspěšně, `texloop` požádá prostřednictvím signálu prohlížeč T_EXového výstupu, aby znovunačetl PDF resp. DVI a obnovil zobrazení. Potom `texloop` přechází znovu do stavu čekání na signály.

Proces `texloop` končí svou činnost buď po obdržení signálu SIGTERM (můžete zmáčknout `Ctrl-C` v terminálu, kde `texloop` běží), nebo ukončením prohlížeče PDF nebo DVI, 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 zalezli 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 T_EXování se vydlábně. Můžeme ovšem potlačit vykonání testu, kterým `texloop` kontroluje stáří zdrojáku a logu. Stačí přidat k parametrům pro `texloop` písmeno A (jako „run T_EX 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 `~/t1pid`, 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 spouštět T_EX zbytečně, ale není to obvyklé. Procesy bez příznaku A (Always) totiž budou T_EXovat jen v případě, že je zdroják novější než log.

Skriptík `texloop` je docela jednoduchý. Má 79 řádků, najdete ho na [1] a můžete si ho přiohnout k obrazu svému.

Příloha: Konečně končím s Emacsem

V následující příloze, která je ovšem delší, než celý článek, rozvedu neformálním způsobem své zkušenosti s textovými editory. Pro lidi, kteří očekávají spíše nezaujaté technické informace a zejména pro fanoušky Emacsu není tato příloha určena. Pro uživatele MS Windows také není tato příloha 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 zkratk. 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. „Odhlečený 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, necht' 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 ovládní, 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 „zmáčkní `Alt` a `Shift`, zmáčkní `A`, pust' `Alt` a stále drž `Shift`, zmáčkní `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 `TEX`ování rovnou z editoru), investoval jsem do něj asi měsíc svého života a pokusil se vytvořit alespoň pár vlastních klávesových zkratk na `TEX`ování a pro aspoň trochu normální ovládní. 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 vedlejší 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 Cěčkových zdrojích a kompilaci nějak rozdýchat, ale s příchodem dalších verzí kompilátoru a novějších knihoven to šlo stále hůř. Takže jsem se rozhodl znovu investovat pár týdnů svého času, ponořit se do zruďné 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 nesprávné p5epnut0 klávesnici. 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, jak vypnout case-insensitive

mód při povelu `isearch`. Zvýrazňování syntaxe pro \TeX ové zdrojáky je v Emacsu svázáno 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`, `gnuzip`, `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 ptání 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 ptání 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 Emacs má 25 krát větší binárku než `joe`. Emacs navíc dynamicky loaduje 43 dalších knihoven, zatímco `joe` loaduje jen základní knihovny a `ncurses`. Editor `joe` dělá přesně to, k čemu byl určen, a dělá to dobře. V souladu se **základním zákonem UNIXu**.

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-nahrad' 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 editor lze přepnout do stavu, kdy edituje ISO soubory a naopak v ISO-terminálu zase mohou editovat UTF8 kódované soubory.
- Neřeší, co řešit nemusí: fonty, přepínání klávesnice. To je starost terminálu.
- Jednoduše přeskočím během editování na místo v souboru, které jsem nedávno opustil a pak se 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 \TeX ových zdrojáků k obrazu svému.
- Editor se naučí slova použitá v otevřeném 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 během chvilky pomocí Google. 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 se mu před tímto rozhodnutím 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 zkratk `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 nespádají 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.

Další možnou nevýhodou joe editoru je jeho až příliš jednoduchá konfigurace maker. Nejsou k dispozici žádné podmínky, žádná práce s proměnnými, jen sled za sebou jdoucích elementárních pokynů pro editor. Abych tedy mohl v joe příjemně T_EXovat, vytvořil jsem si bashový skriptík `texloop`. V [1] najdete kromě zmíněného skriptíku i mou uživatelskou konfiguraci `.joerc`. Můžete se tím inspirovat.

Tak jako dříve v Emacsu, jsem soustředil i v joe editoru T_EXování 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 T_EXovat 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 zpráv o zpracování T_EXu neobtěžuje, když nechci, ale mohu se tam podívat (`Ctrl-KI`).

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 T_EXovat nebo editovat konfigurační soubory vzdáleného serveru. Dělán to od této chvíle stejným editorem s jednotným způsobem ovládní. 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*.

Odkaz

[1] <http://petr.olsak.net/texinunix.html>