Ez aztán a fejlődés!

Cikkünkben a vi egy okosabb változatát mutatjuk be.

A Linuxhoz tartozik egy vim nevű program, mely a vi szerkesztő bővített változata. Nemcsak a vi utánzására képes, hanem szó szerint több száz további szolgáltatást is nyújt, ilyenek például a súgórendszer, a több ablak, a programfordítás, a hibajavítások, a fejlett keresőrendszer és még sorolhatnánk.

Az első lépések

Alapértelmezés szerint a vim vi-megfelelő módban indul el. Ez azt jelenti, hogy az új lehetőségek nagy része nem használható. Bekapcsolásukhoz egy `$HOME/.vimrc` nevű fájlt kell létrehoznunk.
Az *1. listán* egy példafájl látható, melyet úgy is létrehozhatunk, hogy a létező .exrc fájlt lemásoljuk és átnevezzük. Nulla bájt hosszúságú fájllal is működik a dolog.
A vim szerkesztőnek két változata létezik: az egyik, amikor a vim parancs a szerkesztő konzolablakában indul el. Ennél a változatnál abban a terminálablakban folyik a szerkesztés, amelyben a parancsot adtuk ki. A gvim viszont saját ablakot hoz létre. Ez utóbbi módszer hasznosabb, hiszen így a konzolos változatban elérhetetlen menüket és az eszköztárat is használhatjuk.

A súgórendszer

A vim egyik leghasznosabb újítása a bármikor elérhető, beágyazott súgórendszer. A `:help` parancs jeleníti meg a súgóablakot. A parancs után annak a parancsnak a nevét kell beírnunk, melyről adatokat kívánunk szerezni. A `:help /` például a / (keresés) parancsot ismerteti (*1. kép*).
A szövegben a hagyományos szerkesztőparancsokkal (h, j, k, l, Page Up, Page Down stb.) mozoghatunk. Eközben néha az alábbihoz hasonló sorokkal találkozhatunk:

```
/<CR> Search forward the [count]'th latest used
pattern |last-pattern| with latest used |{offset}|.
```

A függőleges vonalak (`|`) közé zárt szöveg vim hiperhivatkozás. Álljunk ezek egyikére (mondjuk a `|last-pattern|`-re), nyomjuk le a Ctrl+] billentyűket,ezzel az adott elemre ugorhatunk.
Az ugrás előtti helyre a Ctrl+T lenyomásával térhetünk vissza.
A súgórendszerből való kilépéshez a vim szokásos kilépés parancsait (`:q` vagy `ZZ`) használhatjuk.

Fájl létrehozása

A vim új lehetőségeinek erejét az alábbi programszöveg beírásával magunk is megtapasztalhatjuk:

```
#include "even.h"
int even(int value)
{
  if (value & 1) == 1)  // Itt hibás a zárójelezés.
       return (1);
  return (0);
}
```

Az első észrevehető újdonság, hogy a szöveg színes. A programlista minden elemtípusa (kulcsszó, karakterlánc, állandó, fordítási elv stb.) más-más színt kap. Ezt a `:syntax on` parancs engedélyezi a .vimrc fájlban. A `:syntax off` kikapcsolja e megkülönböztetést. Egy másik hasznos dolog, hogy nincsen szükség behúzásra (azaz a Tab billentyű nyomogatására a behúzni kívánt sorok elején). Mindezt a `cindent` értéknek köszönhetjük:

1. lista Egy .vimrc példa

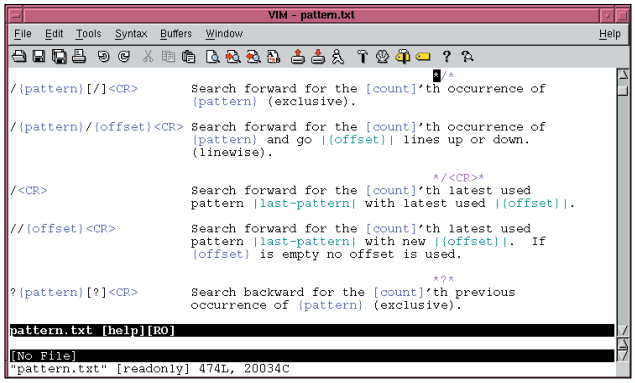
```
:syntax on
" A vimben a sorok összetartozását a \ jelzi.
:autocmd FileType *
\       set formatoptions=tcql nocindent
comments&

:autocmd FileType c,cpp
\       set formatoptions=croql cindent
\       comments=sr:/*,mb:*,ex:*/,://

:set autoindent
:set autowrite

:ab #d #define
:ab #i #include
:ab #b /*************************************
:ab #e **********************/
:ab #l /*-------------------------------------*/

:set shiftwidth=4

:set notextmode
:set notextauto

:set hlsearch
:set incsearch

:set textwidth=70
:set guioptions=-v
```

```
:autocmd FileType c,cpp     :set cindent
```

Ez csak a C és C++ fájlokra vonatkozik. Sajnos, cikkünk terjedelme miatt nem ismertethetem az összes automatikus lehetőséget, de a `:help autocmd` parancs kiadása után mindent megtudhatunk.
Egy megjegyzés: a .vimrc példafájlban található parancsok kicsit összetettebbek, mint a fent említett alakok, de végeredményben ugyanazt művelik.

Visszavonás és ismétlés

Akkor kezdjünk el szórakozni egy kicsit. Álljunk egyik return parancs „r” betűjére és írjuk be, hogy xxxxxx, mire a return eltűnik. Most nyomjuk le az u billentyűt az utolsó változtatás visszavonásához. Ekkor megjelenik az „n”.
A régi vi szerkesztőben csak egyszintű visszavonásra volt lehetőség,

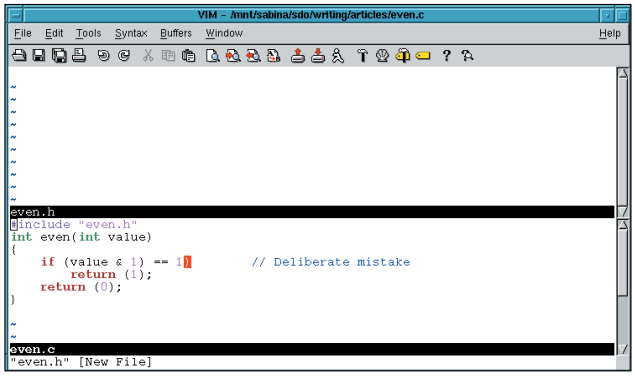


1. kép A vim súgóablaka

a vimben viszont elvileg akárhány módosítást visszavonhatunk. Nyomjunk megint „u”-t, és nicsak, az „rn” is megjelent. Az „u” többszöri begépelésével az egész „return” szót visszavonhatjuk.
És a visszavonás visszavonása? Az is megy: a használandó billentyűk a CTRL+R. Ezt néhányszor ismételve visszavonhatjuk a törlést, mire a „return” fokozatosan ismét eltűnik.

Több ablak

Ha már C fájlt szerkesztünk, akkor fejlécfájlra is szükségünk lehet. Milyen szép is lenne, ha a prototípust egyszerűen bemásolhatnánk a C fájlból a fejlécbe. A vi-jal ezt nem lehet, a vimmel viszont gyerekjáték. Először is töltsük be mindkét fájlt a szerkesztőbe. A `:split even.h` parancs kettévágja az éppen használt ablakot, hatására a felső részbe az even.h, az alsóba pedig az even.c kerül (*2. kép*).

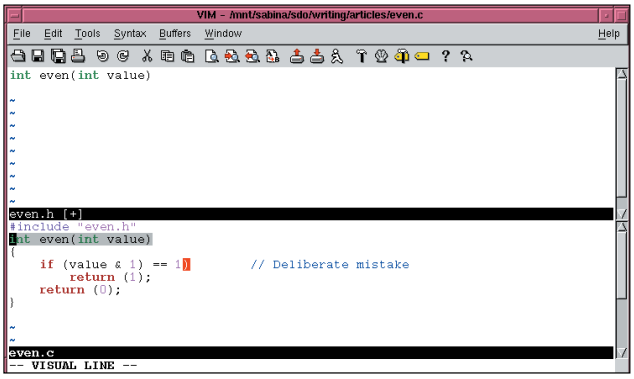


2. kép Egyszerre több ablakban is dolgozhatunk

Ha az alsó ablakból a felsőbe kívánunk átlépni, akkor nyomjuk le a CTRL+W, J billentyűket. A felső ablakra a CTRL+W, K billentyűkkel válthatunk. Az ablak bezárásához a `ZZ` vagy a `:quit` parancsot használhatjuk.

Másolás és beillesztés megjelenítő üzemmódban

A vimnek néhány új üzemmódja is van. Megjelenítő módban a szerkeszteni kívánt szövegrész kijelölése után egy szerkesztőparancsot kell beírnunk. Az alábbi példában az `even` függvény első sorát lemásoljuk, és a felső ablakba illesztjük.
Először is lépjünk az alsó ablakba (CTRL-W J) és álljunk az `even` függvényre. Ezt követően lépjünk át megjelenítő módba a `V` paranccsal. Ekkor az éppen szerkesztett sor kijelölődik, tehát ezt a sort fogja érinteni a beírandó szerkesztőparancs. A nyíl billentyűkkel léphetünk feljebb vagy lejjebb, és eközben a kijelölést is változtatjuk. Ha a kezdőpont fölé lépünk, akkor az afölötti rész jelölődik ki, ilyen egyszerű az egész.
A kijelölés megjelenítő módban soronként történik. Ha a `V` parancsot gépeljük be (karakterenkénti megjelenítő mód), akkor a szöveg minden egyes karakterével külön foglalkozhatunk. A megjelenítő üzemmód harmadik típusa blokkonként dolgozik, és ezt a CTRL-V billentyűkkel indíthatjuk. Ez utóbbi főleg táblázatokkal való munka esetén hasznos, hiszen így az oszlopokat, sorokat külön-külön kezelhetjük.



3. kép Sorok másolása megjelenítő üzemmódban

Akkor tehát a másoláshoz lépjünk soronkénti megjelenítő módba és jelöljük ki a kérdéses sort. Ezt az `y` paranccsal másolhatjuk át a tárolóba. Váltsunk át a felső ablakra, ahol a `p` paranccsal illeszthetjük be az imént másolt szövegrészt. Már csak egy `;` kell a sor végére, és kész is van a fejléc. A *3. kép* az említett parancsokat mutatja működés közben.

Programfordítás

Programunk fordításához egy Makefile is elkélne, ennek megírását azonban most vállalkozó szellemű olvasóimra bízom. Ha a Makefile a helyére került, akkor máris kihasználhatjuk a vim és a make összehangolásának lehetőségét. Mindössze a `:make` parancsot kell kiadnunk a vimből, és a szerkesztő elindítja a make programot és rögzíti annak kimenetét.
Programunkban azonban van egy hiba is:

```
even.c:4 parse error before '=='
```

A vim beolvassa a `:make` parancs kimenetét és látja, hogy a 4. sorban hiba van, tehát erre a sorra állítja a helyőrt, és a hibaüzenetet kiírja a képernyő alsó sorába, még a fájlokat is felcseréli, ha ezt a hiba megjelenítése megköveteli.
Így már igen könnyű a hibajavítás. A következő hibára a `:cn`, az előzőre a `:cp`, az elsőre pedig a `:cc` paranccsal ugorhatunk. Ha a fájl javítását befejeztük, és a következő fájl első hibájára szeretnénk lépni, akkor a `:cnf` parancsot kell használnunk.
A vim tehát igen okos, ha hibakeresésről van szó. Ha a fájl elejéről sorokat törlünk vagy beszúrunk, a vim akkor is emlékezni fog a hibák helyére. Ez a program egyik hatalmas előnye a vi-jal szemben, ahol az első néhány hiba kijavítása után könnyen elcsúszhatott az egész sorszámozás.

Egy használt függvény keresése

A Linux `grep` parancsa nagyszerűen alkalmas azon sorok megtalálására, melyben egy bizonyos függvény vagy annak meghatározása szerepel. Egyszerűen adjuk ki az alábbi parancsot:

```
$ grep -n even *.c
```

Ennek hatására a könyvtárban található összes `.c` kiterjesztésű fájl minden olyan sora megjelenik, mely tartalmazza az `even` szót, és a sor elején közli a sorszámot is.
A vimben is van egy `:grep` parancs, működése nagyon hasonlít

a `:make`-re. A Linux grep parancsát futtatja, figyeli a kimenetet és a megtalált helyek között a `:cn`, `:cp`, `:cc` és `:cnf` parancsokkal lépkedhetünk, akárcsak a `:make` esetében.

A behúzások javítása

Ha a `cindent` értéket bekapcsoljuk, a vim új szöveg beszúrása esetén megfelelően behúzza a sorokat. De mi a helyzet a már létező szöveggel? Itt lép be a képbe a vim `=` parancsa.
Ez a parancs egy szövegtömbön futtatja le a vim belső, a behúzást vezérlő programját de az equalprog lehetőséggel a feladatot akár egy külső programra is rábízhatjuk.
Nézzük, miként működik mindez egy szabványos C kódrészlet esetében. Az `=` parancsot kétféleképpen hívhatjuk meg. Az első módszer, hogy az `= {motion}</>` parancsot használjuk, a második, hogy előbb megjelenítő üzemmódba lépünk, kijelöljük a szövegrészt, majd kiadjuk az `=` parancsot.
A rész behúzásának átállítását kezdhetjük azzal is, hogy a szakasz első `{` jelére állunk. Most írjuk be az `=%` parancsot. Ezzel a vimet arra utasítjuk, hogy innentől kezdve húzza be a szöveget egészen addig, ahol a második parancsot eléri. Jelen esetben a második parancs a `%`, ennek hatására a vim a záró `}` jelre ugrik.
Ha mindezt megjelenítő módban kívánjuk elvégezni, akkor álljunk az első `{` jelre, majd lépjünk soronkénti megjelenítő módba a `V` paranccsal. Ezután álljunk a a záró `}` jelre, ehhez bármilyen ugróutasítást használhatunk, majd az `=` paranccsal húzzuk be a kijelölt részt.

Keresés

A vimnek akad néhány érdekes keresési lehetősége is. Az első ezek közül a szűkítő (más szóval növekményes) keresés, ezt a `:set incsearch` kapcsolja be. Kereséskor alapértelmezés szerint az egész parancsot be kell írnunk (például az include keresésekor a `/include`-t). A szűkítő keresés használata során azonban már az `i` beírása után az első i-vel kezdődő szóra ugrunk, az `n` után az első in-nel kezdődő szóra stb. Ahogy a szót karakterenként felépítjük, úgy közeledünk a keresett szóhoz.
Egy másik fontos újítás a kiemelő keresés (hlsearch). Ez annyit jelent, hogy a vim kiemeli a megtalált kifejezést, nemcsak az első előfordulást, hanem mindegyiket. Ez a lehetőség hasznos lehet például félregépelések megtalálásában, hiszen a vim a hibás szó minden előfordulását kiemeli. A kiemeléseket a `:nohl` paranccsal átmenetileg – a következő keresőparancs begépeléséig – ki is kapcsolhatjuk.
A vim a keresésekről adatbázist is készít. Tegyük fel, hogy már jó néhány keresést indítottunk és szeretnénk visszatérni egy régebbi eredményéhez. Ilyenkor a `/` paranccsal indítsunk egy keresést, majd a Fel, illetve Le billentyűkkel lépkedhetünk a régebbi keresések között.

Billentyűzetmakrók

A vim egyik leghatékonyabb parancsa a `.` (pont), mely az utolsó szerkesztési műveletet ismétli meg. E parancs azonban nem nyújt minden esetben megfelelő megoldást, hiszen csak egyetlen parancsot ismétel. De mi történik akkor, ha valami összetettebb feladatot szeretnénk elvégezni? Ekkor használhatjuk a billentyűzetmakrókat. Segítségükkel több parancsot rögzíthetünk egy tárolóba, majd ezeket végrehajthatjuk.
Hogy jobban megértsük a makrók működését, nézzünk most egy példát. Tegyük fel, hogy a következő sorokat kívánjuk módosítani:

```
stdio.h
time.h
unistd.h
stdlib.h
```

Mondjuk az a vágyunk, hogy ez a négy kifejezés #include alakra módosuljon (például `#include <stdio.h>`).

Egy megjegyzés RedHat-felhasználóknak

Alapértelmezés szerint a RedHat egy vim-minimal nevű csomagot telepít, mely a vim lecsupaszított változata. Ebből egy rakás hasznos szolgáltatást kihagytak, például az írásmód szerinti színkiemelést. A szerkesztő teljes változatának használatához telepítsük az alábbi csomagokat:

vim-X11-*változat*.i386.rpm
vim-common-*változat*.i386.rpm
vim-enhanced-*változat*.i386.rpm

A **változat** a rendszerhez kapott vim változatszáma.

Kezdjük a `qa` parancs beírásával, ennek hatására az ezt követő parancsok az `a` tárolóba kerülnek. A tárolókat az ábécé betűivel jelölhetjük. Most végezzük el a szerkesztést. Ugorjunk a sor elejére (`^`), illesszük be az `#include` fordítási elvet, majd tegyük `<` és `>` jelek közé a fájlnevet. A `q` paranccsal állítsuk le a makrórögzítést. A szöveg most így néz ki:

```
#include <stdio.h>
time.h
unistd.h
stdlib.h
```

A helyőr most a második soron áll. A makró futtatásához a `@a` parancsot használjuk, ennek eredményeképpen a fájl az alábbiak szerint változik:

```
#include <stdio.h>
#include <time.h>
unistd.h
stdlib.h
```

A `@a` parancs elé számot írva megadhatjuk, hogy a makró hányszor fusson le. A `2@a` kiadása után tehát ezt kapjuk:

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <stdlib.h>
```

A program tartalomjegyzéke

A vimhez kapott `ctags` parancs segítségével C fájlok tartalomjegyzékét készíthetjük el, ezt a vim „tags” fájlnak nevezi. Az alábbi parancs hatására létrejövő tags állomány a munkakönyvár C fájljaiban található összes függvény helyét tartalmazza:

```
ctags *.c
```

A létrehozott tags fájl rendkívül hasznos segítséget jelent a C programozók számára.
Tegyük fel, hogy egy fájlt szerkesztünk és az `olvas_bekezdes` függvényt keressük. A kódot böngészve rájövünk, hogy az `olvas_bekezdes` az `olvas_mondat` függvényt hívja meg. Szeretnénk megtudni, hogy ez a függvény mit csinál. Ha a függvény nevére állunk és a Ctrl+] billentyűket lenyomjuk, a vim ezen függvény meghatározásához ugrik. Itt pedig az derül ki, hogy az `olvas_mondat` az `olvas_szo` függvényt használja, tehát most erre állunk, és itt nyomjuk le a Ctrl+] billentyűket, hogy a függvényhez ugorjunk.

A vim a tagverem segítségével kíséri figyelemmel, hogy eddig mely helyeken jártunk. Ha azt szeretnénk megállapítani, hogy hol járunk, ebben a listában használjuk a `:tags` parancsot.

```
:tags
# TO tag               FROM line  in file/text
1  1 olvas_mondat      3  olvas_mondat();
2  1 olvas_szo         8  olvas_szo();
>
```

Ebben a példában az `olvas_szo` függvényről (ez nem szerepel a listában) először az `olvas_mondat`-ra, majd az `olvas_szo`-ra ugrottunk. Ha egy lépéssel szeretnénk visszaugrani, akkor a CTRL+T parancsot használjuk.

Kifinomult ugrási műveletek

Ha a CTRL+] paranccsal egy tagra lépünk, akkor az egész képernyő ugrik. Ha ehelyett a CTRL+W CTRL+] billentyűket nyomjuk le, akkor az éppen használt ablak kettéválik, és az ugrás az új ablakban történik.

Interaktív ugrás

Tegyük fel, hogy öt-hat kis program található a munkakönyvtárban, és az egyik `main` függvény helyét keressük. Ha a

```
:tag main
```

parancsot adjuk ki, a szerkesztő a `main` első előfordulására ugrik, és nem biztos, hogy ezt szeretnénk.
A `:tselect` parancs az adott névhez illeszkedő összes tagot megjeleníti, s ezek közül mi magunk választhatjuk ki a megfelelőt. Például:

```
:tselect main
# pri kind tag                  file
> 1 F C f    main                 a_test.cpp
       int main{ int argc, char* argv[] )
2 F    f       main                acp.cpp
       int main( int argc, char* argv[] )
3 F    f       main                add.cpp
       int main(int argc, char* argv[])
Enter nr of choice (<CR> to abort): 3
```

Mi történik, ha a névnek csak egy részét ismerjük, például a *valami*-process-*valami*-data nevű függvényre szeretnénk ugrani? A `:tags` parancs szabványos kifejezéseket is elfogad. Ebben az esetben a

```
:tag /process.*data
```

paranccsal az első ilyen sorra ugorhatunk. Ha a megadott kifejezésre több függvény is illeszkedik, akkor a `:tselect` segítségével választhatunk közülük.

Sortörés

A programok kódot és megjegyzéseket tartalmaznak. A kódnak pedig saját szerkezete van. Nincs szükségünk olyan szerkesztőre, mely a kód írásakor sortörést használ, viszont a megjegyzések hagyományos szövegek, ezeknél nyugodtan használhatjuk e lehetőséget. Sőt, a legtöbb esetben kifejezetten előnyös, ha a hosszú szöveges sorokat megtörjük.
A vim szerkesztő képes elkülöníteni a kódot a megjegyzésektől. Beállíthatjuk, hogy csak a megjegyzések esetében végezzen sortörést, a kódot pedig hagyja változatlanul. Ehhez az alábbi módosításokat kell elvégeznünk a .vimrc fájlban:

```
:autocmd FileType c,cpp
\       set formatoptions=croql
\       cindent
\       comments=sr:/*,mb:*,ex:*/,://
```

(Megjegyzés: a sorok folytonosságát a `\` jelzi)
Az `:autocmd` parancs arra utasítja a vimet, hogy az alatta elhelyezett parancsokat akkor hajtsa végre, ha C vagy C++ fájllal dolgozunk (ezt a .c és a .cpp kiterjesztésből állapítja meg).
A `formatoptions` hatására a megjegyzéseknél sortörést hajt végre, a kód viszont változatlan marad. A

```
set comments= sr:/*,mb:*,ex:*/,://
```

sor tudatja a vimmel, hogy a megjegyzések hogyan néznek ki. Itt a C nyelvben (`/*` és `*/`) és a C++-ban (`//`) megszokott megjegyzésjeleket állítottuk be. Azt is elmagyaráztuk a vimnek, ha egy C-stílusú megjegyzés közepén járunk, akkor minden sort `*` karakterrel kezdjen. Ezen értékek beállítása után, ha `/*`-t írunk és ENTER-t nyomunk, a vim a következő sor elejére magától kiteszi a `*` jelet.

Összegzés

Jelen esetben nem áll módomban hosszasan elemezni e lehetőséget, de remélem annyi kiderült, hogy egy igen hasznos szolgáltatásról van szó, mellyel érdemes eljátszadoznunk egy kicsit. Aki részletesebb tájékoztatásra vágyik, az olvassa el a beépített súgót. A vim egy rendkívül rugalmas szerkesztő, igyekeztem bemutatni néhány érdekesebb lehetőségét, de ezzel is csupán a felszínét érintettem. Több száz vagy akár ezer olyan parancs létezik a vimben, amit itt meg sem említettem. Ezekről a szerkesztővel kapott leírásban vagy a beépített súgóban olvashatunk.
Bízom benne, hogy a vim néhány új lehetőségének ismertetésével sokakat sikerült bevezetnem a hatékonyabb szerkesztés világába. Hogy innen merre haladunk, az csupán rajtunk múlik.

Steve Qualline
jelenleg San Diegóban él és a Nokia mobiltelefonokkal foglalkozó részlegénél dolgozik, hétvégeken pedig igazi mozdonyvezető a californiai Polwayben található turistavasútnál.

Ötlet

Mozgás a vi-ban

A vi-ben a (,), [,], { és } jelek között is mozoghatunk: a % parancs segítségével a jelpár záró tagjára ugrunk. Nézzünk egy példát:

```
( ezmegaz [ amaz
itt egy másik sor
{  ]
} )
```

Ha itt az első kapcsos zárójelre állunk, a % paranccsal a jelpár záró tagjára, azaz egy sorral lejjebb ugorhatunk. Ezt a módszert használhatjuk megjegyzésekben is, és így gyorsan mozoghatunk előre-hátra terjedelmes programjainkban. A C-hez hasonló nyelvekben pedig kitűnő eszköz a függvények kezdetének és végének betájolására.