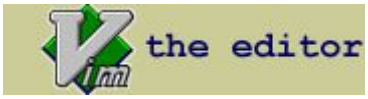
Viola Zoltán
violazoli@gmail.com

Bevezetés a VIM szövegszerkesztő kezelésébe

(A Vim ugyanis sokkal többet tud, mint ami e könyvben le van írva!)

Verzió: **1.14**



Tartalom

Bevezetés

Ez a könyvecske a VIM nevű szövegszerkesztő kezelését lenne hivatott bemutatni. A Vimről eltérő véleményeket hallani: egyesek imádják, mások szerint olyan bonyolult, hogy használata „agykárosodást okoz". Az én véleményem az, hogy egyáltalán nem bonyolult, mindössze eleinte egy kissé szokatlan. Már amiatt sem lehet bonyolult, mert eredetileg *titkárnők* számára találták ki, vagyis olyasvalakik számára, akikre nem jellemző a magas szintű számítástechnikai szakismeret!

E mű természetesen nem tér ki a Vim minden képességére, mindenesetre (ennek ellenére) szándékaim szerint az eddigi legbővebb leírás, mely magyar nyelven elérhető ezidőszerint. Törekedtem a hibamentes leírásra, ugyanakkor nem vállalok semmiféle felelősséget az esetleg mégis bennmaradó hibákért, azaz az itt leírtakat mindenki a saját felelősségére használja!

Jószándékú kritikát és bővítésre tippeket, trükköket, ötleteket szívesen fogadok az email-címemre.

A Vim a Vi szövegszerkesztő továbbfejlesztett változata – a neve is ezt jelenti angolul: ***VI*** *i* ***M****proved*.

Sok platformra elérhető, Windowsra is, Linux és Unix alatt pedig afféle „alap felszerelés". Ez is egyik oka annak, amiért érdemes megtanulni: ha semmi más szövegszerkesztő nincs is valami disztribúcióban, Vim valószínűleg akkor is akad. De Vim van AmigaOS, Atari MiNT, BeOS, DOS, MacOS, NextStep, OS/2, OSF, RiscOS, SGI, VMS, Win16 + Win32 (Windows95/98/00/NT) és FreeBSD alatt is...

A másik ok pedig amiért érdemes megbarátkozni vele, az a hatékonysága. Számtalan billentyűzetparancsa van, amik megkönnyítik és gyorssá teszik a munkát, de akár mi is írhatunk bele makrókat, sőt, pluginezhetjük is.

Mindazonáltal nem tagadható, hogy a Vim nem egy úgynevezett „intuitív" szövegszerkesztő. Manapság a számítógépfelhasználók hozzászoktak az olyan szövegszerkesztőkhöz, melyeket – kis túlzással mondva - „a hülye is tud kezelni". Ilyen az MS Office, a LibreOffice... de a Vim, az nem ilyen. Többet tud mint amazok, de csak azt szolgálja, aki hajlandó erőfeszítést tenni az elsajátítására. A Vim olyan eszköz, melynek a kezelését meg kell tanulni.

Nos, ebben a tanulásban próbál segíteni e könyv.

1. rész

Alapparancsok

A Vim indítása

A Vim-et a **vim filenév** paranccsal indíthatjuk, vagy, ha csak olvasásra akarunk megnyitni egy állományt, akkor a **view filenév** paranccsal.
Egyéb lehetőségek:
vim +1000 valami.conf - az 1000. sornál nyitja meg a valami.conf fájlt
vim +/log - a „log”-szórészt tartalmazó sornál nyitja meg a fájlt
vim -b +/"Continue" filenév - Bináris fájlok szerkesztése, a példában a „Continue” szónál nyitja meg a fájlt, átírhatjuk (de csak az eredeti hosszal, azaz ez esetben 8 karakterrel, különben segfault)

Több fájl megnyitása:

vim file1 file2 file3

Indulás után az első fájlt látjuk. Mindegyik file külön úgynevezett *bufferbe* kerül. (A bufferek kezeléséről részletesen lásd a megfelelő fejezetet). A következő parancsokat lehet használni a fájlok közti váltáshoz:

:n	ugrás a következő fájlra
:bp	ugrás az előző fájlra
Ctrl-^	váltás az utolsó két fájl között
:rew	vissza az első fájlra
:arg	alul megmutatja a nyitott fájlok nevét, az aktuális szögletes zárójelek közt

vim -p file1 file2 file3 - ez is megnyit egyszerre több fájlt, de mindegyiket külön úgynevezett „tab”-on – a file1 kerül az első tabra, a file2 a másodikra, s a file3 a harmadikra. A tabok használatáról részletesen lásd a megfelelő fejezetet.

A Vim üzemmódjai

A Vimnek egy, két vagy három üzemmódja van. Hogy pontosan mennyi, az ugyanis attól függ, mit értünk „üzemmód” alatt. Egy bizonyos: míg az ismertebb, „normális” szövegszerkesztők alapesetben mindig *szöveget* várnak el bevitelre, addig a Vim *parancsot.* A parancs egy vagy egynéhány betű sorozata, esetleg számokkal kiegészítve (például **d$**). Van természetesen olyan parancsa is, nem is egy, ami által be lehet vinni szöveget. Ám a szövegbevitel végén le kell nyomnunk az Esc gombot, ebből tudja meg a Vim, hogy itt fejeztük be a szöveg bevitelét. Tehát a szövegbevitel is parancs a Vim esetében. Mindazonáltal egyes Vim-leírások a szövegbevitelt külön üzemmódnak - „inzert mód” - szokták tartani. Ennek lenne ellentéte az úgynevezett „parancs mód”, más néven „normál mód”. Lényeg, hogy ha nem tudod milyen „üzemmódban” vagy, akkor nyomd meg az Esc-t addig, míg a Vim nem kezd el sípolni, ekkor biztosan „parancs módban” vagy.

Másik „üzemmódja” az úgynevezett „parancssor”. A Vimnek van ugyanis parancssora is, a képernyő legalján. Ide a kettőspont (:) megnyomásával jutunk el, itt aztán be kell gépelnünk a parancsot, amit Enterrel kell lezárnunk. Egy példa rá:

:colorscheme desert

- ez a parancs átvált a „desert” nevű színsémára.

Ha véletlenül nyomtál kettőspontot, az sem tragédia, a parancssorból is kiszállhatsz a közönséges „parancs módba” az Esc megnyomásával.

Kilépés a Vimből és az állományok mentése, megnyitása

:q - kilép
:q! - kilép mentés nélkül, akkor is ha módosítottad a fájlt
:w - ment
:qa! - kilépés mentés nélkül minden ablakból (ha több ablak van nyitva)
:w! - csakazért is ment, akkor is, ha csak olvasásra nyitottuk meg a fájlt, vagy ha csak olvasható, de van írásjogunk
:wq - ment, majd kilép
:x - ment ha kell, majd kilép
ZZ - ment ha kell, majd kilép
ZQ - ugyanaz mint :q!
:sav filenév - mentés másként „filenév” néven, és azt megnyitja
:e . - könyvtárstruktúra megnyitása, tallózásra
:E - az aktuális könyvtár tallózása
:e filenév - megnyitja a fájlt
:w újnév - elmenti az állományt „újnév” néven

Navigáció a szövegben

h – mozgás a szövegben egy karaktert balra
j – mozgás a szövegben egy sort le
k – mozgás a szövegben egy sort fel
l – mozgás a szövegben egy karaktert jobbra

de általában működnek a kurzorbillentyűk is. Ha mégsem, e leírásban később mutatok példát arra is, hogyan lehet azt megcsinálni. Annak oka, hogy a h,j,k,l billentyűkkel (is) lehet mozogni a szövegben, az, hogy a régi terminálokon nem voltak külön kurzormozgató gombok.

gg - a dokumentum elejére ugrik
G - a dokumentum utolsó sorának elejére ugrik
:$ - a dokumentum utolsó sorának elejére ugrik
^ - az első nem „whitespace” karakterre ugrik
0 - (nulla) a sor elejére ugrik
$ - a sor végére ugrik
w - egy szót előre ugrik
b - egy szót hátra ugrik
{ - ugrás a bekezdés elejére
} - ugrás a bekezdés végére
Megjegyzés: **a Vim szerint a bekezdéseket egy üres sor választja el egymástól!**
(- ugrás a kurzort tartalmazó mondat elejére. Ha a kurzor a mondat első karakterén áll, akkor az előző mondat elejére ugrik.
) - ugrás a következő mondat első karakterére.
Megjegyzés: a Vim szerint egy mondat olyan karaktersorozat, amit a „.” (pont), „?” vagy „!” fejez be, és ezt egy vagy több szóköz, Tab vagy újsor követi.

% - ha zárójelen állunk, átugrik a párjára (nyitó zárójelről csukóra és fordítva)
fu - előre az első „u” betűig (soron belül)
Fu - vissza az utolsó „u” betűig (soron belül)
ti - előre az első „i” előtti karakterig
Ti - vissza az utolsó „i” utáni karakterig
:300 - a 300. sorhoz ugrik
300G - a 300. sorhoz ugrik
10| - a 10-edik oszlophoz ugrik a soron belül
***** - átugrik a következő szóra, ami megegyezik azzal, amin álltál amikor a *-ot nyomtad
# - visszaugrik az előző szóra, ami megegyezik azzal, amin álltál amikor a *-ot nyomtad
Ctrl-f - egy oldalt előre
PageDown - egy oldalt előre
Ctrl-b - egy oldalt vissza
PageUp - egy oldalt vissza

Ctrl-e - gördítés le (vagy fel, nézőpont kérdése)

Ctrl-y - gördítés fel (vagy le, nézőpont kérdése)
zz - a sor amiben a kurzor áll az ablak közepén lesz
zt - a sor amiben a kurzor áll az ablak első sora lesz; t mint *top*
zb - a sor amiben a kurzor áll az ablak utolsó sora lesz; b mint *bottom*
M - kurzort az ablak közepére
L - kurzort az ablak aljára
H - kurzort az ablak tetejére

A Vimben a legtöbb parancshoz meg lehet adni, hogy hányszor hajtódjon végre, így lehet még az előzőknél is ötletesebben mozogni:
3j - 3 sort le
4h - 4 karaktert vissza
5k - 5 sort fel
6l - 6 karaktert előre
3Ctrl-f - 3 oldalt előre
3Ctrl-b - 3 oldalt vissza

Ha egy sor első karakterén állunk, s megnyomjuk a „kurzorbalra” gombot, a Vim nem ugrik az előző sor utolsó karakterére, hanem sípol. Hasonlóképp, ha egy sor utolsó karakterén állunk, s megnyomjuk a „kurzorjobbra” gombot, a Vim nem ugrik a következő sor első karakterére, hanem sípol. Ezt elkerülendő, hogy az elterjedtebb szövegszerkesztőkhöz hasonlóan viselkedjék e tekintetben, vegyük fel a .vimrc fájlba a következő két sort:
set whichwrap+=<
set whichwrap+=>
Ezután (a Vim újraindítása után) már a kívánt módon fog viselkedni.

Keresés

/valami - „valami” keresése a szövegben előre
?valami - „valami” keresése a szövegben hátrafelé (visszafelé)
a „valami” helyére úgynevezett „szabályos kifejezést” (regular expression, röviden „regexp”) is írhatunk, ami végtelen távlatokat nyit meg a gondolkodó ember előtt...A szabályos kifejezésekről részletesen lásd a megfelelő fejezetet.
n - ugrás a következő találatra (*next*)
/ / - ugrás a következő találatra (*next*)
N - ugrás az előző találatra
?? - ugrás az előző találatra
/<Ctrl-r><Ctrl-w> - vagyis ha a „/” után megnyomjuk a Ctrl-r és a Ctrl-w billentyűket, akkor a „keresés mezőbe”) (a parancssorba) behúzza azt a szót, amin a kurzor épp áll.
/<Ctrl-r>" - vagyis ha a „/” után megnyomjuk a Ctrl-r és az " (idézőjel) billentyűket, akkor a „keresés mezőbe” (a parancssorba) behúzza az utoljára másolt szöveget.
/\<jó\> - megkeresi a különálló „jó” szavakat, de NEM találja meg a „jó” stringet szavak belsejében, például a „hajó”-ban.
/abc\n*def - megkeresi azt a szöveget, ami úgy kezdődik hogy „abc”, majd utána nulla vagy több újsor következik, majd „def”.

Egyéb keresési patternek:
\n újsor karakter (sorvég)
_s whitespace (space vagy tab) vagy újsor karakter
_^ sor kezdete (zéró)
_$ sorvég (zéró)
_. bármely karakter, beleértve az újsort is

A fenti patterneket nemcsak a keresés parancsnál lehet használni, de a csere parancs esetében (**:s**) is.

/piros\|zöld\|kék - mindhárom szóra rákeres, tehát jelzi a „piros”, a „zöld” és a „kék” összes találatát. (A „**\|** ” pattern „vagy” kapcsolatot képez).

Megjegyzendő, hogy a Vim a kurzort az első találat első karakterére pozicionálja. Ha azonban így adjuk ki a keresési parancsot:
/valami/e
akkor a találat utolsó karakterére kerül a kurzor.

A keresési találatokat a Vim kiszínezi. Ez nagyon hasznos, de ha már nem kellenek, illene tudni kikapcsolni a színezést. Ez elérhető a
:nohlsearch
paranccsal. Legcélszerűbb ennek hosszas begépelése helyett egy gyorsbillentyűt kreálni rá, azaz vegyük fel a következő sort a **.vimrc** fájlba:
map <F4> :nohlsearch<CR>
Ennek hatására (a Vim újraindítása után) az **F4** gomb lenyomása megszünteti a találatok kijelzését.
Csak meghatározott sorokban keresés:

/\%>199l\%<300llimit

A fenti példa megkeresi a „limit” szó előfordulásait a 199-edik és a 300-adik sorok közti tartományban. Amint látható, a kezdősort a **\%>kezdősorszáml** míg az utolsó sort a **\%<végsorszáml** módon kell megadni. A „kezdősorszám”-adik és „végsorszám”-adik sorban nem keres! Ha tehát az első sorban is akarunk keresni (a dokumentum legelső sorában) akkor a \%>1l HELYETT a **\%>0l** alakot kell használnunk!

Szövegbevitel és szerkesztés

i - szöveg beszúrása a kurzor elé (insert)
a - szöveg beszúrása a kurzor mögé (append)
I - szöveg beszúrása a sor elejére (a kurzor a sor első nem whitespace karakterére kerül, és a Vim inzert módba vált)
A - szöveg beszúrása a sor végére (a kurzor a sor végére ugrik, és a Vim inzert módba vált)

Amikor „inzert" üzemmódban vagyunk, a Vim a képernyő aljára kiírja, hogy
-- INSERT --
No most inzert üzemmódban rendelkezésünkre állnak bizonyos szövegbevitelt megkönnyítő funkciók. Íme:

Ctrl-p - kiegészít, ha már volt úgy induló szó
Ctrl-x Ctrl-l - a teljes sort kiegészíti, ha már volt úgy induló sor
Ctrl-x Ctrl-f - filenév kiegészítés az aktuális könyvtárból
Ctrl-v - ha a Ctrl-v után nyomunk meg egy speciális billentyűt (például az Esc-t) annak Ascii kódja közvetlenül kerül bevitelre.

Egyéb szerkesztési parancsok:
o - új sor nyitása a kurzor sora alatt
O - új sor nyitása a kurzor sora fölött.
daw - kurzorpozíciótól függetlenül, ha a szón belül vagyunk, kitörli azt, az utána való whitespace karakterekkel együtt
diw - kurzorpozíciótól függetlenül, ha a szón belül vagyunk, kitörli azt, az utána való whitespace karakterek nélkül (azok tehát megmaradnak)
da" - kurzorpozíciótól függetlenül, ha idézőjelek által közrezárt szövegrészen belül vagyunk, kitörli azt, az idézőjelekkel együtt
di" - kurzorpozíciótól függetlenül, ha idézőjelek által közrezárt szövegrészen belül vagyunk, kitörli azt, az idézőjelek azonban megmaradnak
da(- kurzorpozíciótól függetlenül, ha kerek zárójelek által közrezárt szövegrészen belül vagyunk, kitörli azt, a zárójelekkel együtt
di(- kurzorpozíciótól függetlenül, ha kerek zárójelek által közrezárt szövegrészen belül vagyunk, kitörli azt, a zárójelek azonban megmaradnak
da[- kurzorpozíciótól függetlenül, ha szögletes zárójelek által közrezárt szövegrészen belül vagyunk, kitörli azt, a zárójelekkel együtt
di[- kurzorpozíciótól függetlenül, ha szögletes zárójelek által közrezárt szövegrészen belül vagyunk, kitörli azt, a zárójelek azonban megmaradnak
da{ - kurzorpozíciótól függetlenül, ha kapcsos zárójelek által közrezárt szövegrészen belül vagyunk, kitörli azt, a zárójelekkel együtt
di{ - kurzorpozíciótól függetlenül, ha kapcsos zárójelek által közrezárt szövegrészen belül vagyunk, kitörli azt, a zárójelek azonban megmaradnak
da< - kurzorpozíciótól függetlenül, ha **< >** jelek által közrezárt szövegrészen belül vagyunk, kitörli azt, a határoló **<** és **>** jelekkel együtt
di< - kurzorpozíciótól függetlenül, ha **< >** jelek által közrezárt szövegrészen belül vagyunk, kitörli azt, a határoló **<** és **>** jelek azonban megmaradnak

dap - kurzorpozíciótól függetlenül, törli az aktuális bekezdést
das - kurzorpozíciótól függetlenül, ha a mondaton belül vagyunk, kitörli azt, az utána való whitespace karakterekkel együtt
2das - kurzorpozíciótól függetlenül, ha a mondaton belül vagyunk, kitörli azt, az utána való whitespace karakterekkel együtt, s kitörli a következő mondatot is.
dis - kurzorpozíciótól függetlenül, ha a mondaton belül vagyunk, kitörli azt, az utána való whitespace karakterek nélkül
yap - kurzorpozíciótól függetlenül, kimásolja az aktuális mondatot a Vim vágólapjára, amit aztán a **p** vagy **P** parancsokkal beilleszthetünk. A kurzort a mondat első karakterére mozgatja.
%d - a fájl tartalmának törlése
:r filenév - a megadott nevű fájl tartalmát beilleszti a kurzor pozíciójához

A billentyűzeten nem szereplő karakterek Ascii kódját közvetlenül bevihetjük a Vimbe a következő módon:
Inzert módban nyomjuk meg a **Ctrl-v** billentyűkombinációt. Erre a Vim kiír egy **^** jelet a kurzor aktuális pozíciójához. Most billentyűzzük be a karakter Ascii kódját. Ha az 3 decimális számjegyből áll, a Vim rögtön kicseréli a **^** karaktert a szám által reprezentált karakterre. Ha kevesebb mint 3 számjegyből áll a kód, akkor csak nyomjuk meg a kívánt karakter után a bevinni kívánt szövegben következő karaktert, s a Vim mindkettőt beírja.

Törlés és átírás

x - a kurzor alatti karakter törlése
Del - a kurzor alatti karakter törlése
X - a kurzor előtti karakter törlése
Backspace - a kurzor előtti karakter törlése
3x - 3 karakter törlése
D - törlés a sor végéig
dd - a teljes sor törlése
5dd - 5 sor törlése

A törlésre alapvetően a **d** (*delete*) parancsot használjuk. Önmagában ez nem működik, csak úgy, hogy a „d" billentyű után lenyomunk még valamely másik billentyűt is, mely a mozgás irányára utal. A Vim egyik legszebb tulajdonsága ugyanis, hogy a parancsokat kombinálni lehet egymással, például a törlést a mozgással. Íme a példák erre:

d0 - törlés a sor elejéig
d$ - törlés a sor végéig, ugyanaz mint a **D**
de - szó törlése előrefelé (a szó végéig töröl)
dw - szó törlése előrefelé (a következő szó elejéig töröl)
db - szó törlése hátrafelé (a szó elejéig töröl)
dG - törlés a dokumentum végéig
dgg - törlés a dokumentum elejéig
3dw - a következő 3 szó törlése
d} - törlés a bekezdés végéig
d{ - törlés a bekezdés elejéig
das - a kurzorpozíciótól függetlenül, kitörli a teljes mondatot
d) - törlés a mondat végéig
d(- törlés a mondat elejéig
d/valami - törlés előre, amíg „valami"-ig nem érünk (valami megmarad)
d?valami - törlés visszafelé, „valami"-t is beleértve

:g/valami/d - minden sor törlése, amiben van „valami"
:v/valami/d - minden sor törlése, amiben nincs „valami"

Átírás

A **d** párja, a **c** (change), szolgál az átírásra. A **c** ugyanaz, mintha a megfelelő **d**-s parancs után nyomnál egy **i**-t, azaz töröl, majd utána inzert módba rak.

c0 - a kurzor és a sor eleje közti szöveg átírása
C - a kurzor és a sor vége közti szöveg átírása
c$ - a kurzor és a sor vége közti szöveg átírása
ce - átírás a szó végéig (a szó végéig töröl)
cw - átírás a szó végéig (a következő szó elejéig töröl)
cb - a szó átírása hátrafelé (a szó elejéig töröl)

3cw - a következő 3 szó átírása
c} - átírás a bekezdés végéig
c{ - átírás a bekezdés elejéig
c/valami - átírás előre, amíg „valami”-ig nem érünk (valami megmarad)
c?valami - átírás visszafelé, „valami”-t is beleértve
r - a kurzor alatti karakter cseréje egy másikkal (*replace*)
s - a kurzor alatti karakter helyettesítése bármennyi karakterrel (*substitute*, inzert módba tesz)
3s - a következő három karakter helyettesítése bármennyi karakterrel (inzert módba tesz)

Kivágás, másolás és beillesztés

A **kivágás** lényegében nem más, mint a „**d**” parancs, amit ugyanis azzal törlünk, az a vágólapra kerül. A **másolás** az „**y**” (*yank*) paranccsal lehetséges, Pontosan úgy kell használni, mint a **d**-t.

yy - az egész sor a vágólapra
Y - az egész sor a vágólapra
6yy - 6 sor a vágólapra
y0 - kurzortól a sor elejéig a vágólapra
y$ - kurzortól a sor végéig a vágólapra
yw - szó másolása előre (a kurzortól a következő szó elejéig a vágólapra)
ye - szó másolása előre (a kurzortól a szó végéig a vágólapra)
y/valami - a kurzortól előre „valami”-ig a vágólapra
y?valami - a kurzortól visszafelé „valami”-ig a vágólapra
y10h - az előző 10 karakter másolása a vágólapra
y) - másolás a mondat végéig a vágólapra
y(- másolás a mondat elejéig a vágólapra
yas - a kurzor pozíciójától függetlenül, a teljes mondatot a vágólapra másolja
ya(- a kurzor pozíciójától függetlenül, a teljes, kerek zárójelek által határolt blokkot a vágólapra másolja
ya[- a kurzor pozíciójától függetlenül, a teljes, szögletes zárójelek által határolt blokkot a vágólapra másolja
ya{ - a kurzor pozíciójától függetlenül, a teljes, kapcsos zárójelek által határolt blokkot a vágólapra másolja
ya< - a kurzor pozíciójától függetlenül, a teljes, **<** és **>** jelek által határolt blokkot a vágólapra másolja
yi(- a kurzor pozíciójától függetlenül, a teljes, kerek zárójelek által határolt blokkot a vágólapra másolja, de a határoló zárójeleket nem
yi[- a kurzor pozíciójától függetlenül, a teljes, szögletes zárójelek által határolt blokkot a vágólapra másolja, de a határoló zárójeleket nem
yi{ - a kurzor pozíciójától függetlenül, a teljes, kapcsos zárójelek által határolt blokkot a vágólapra másolja, de a határoló zárójeleket nem
yi< - a kurzor pozíciójától függetlenül, a teljes, **<** és **>** jelek által határolt blokkot a vágólapra másolja, de a határoló **<** és **>** jeleket nem

Beillesztés
p - (kis „p” betű) vágólap tartalmának beillesztése a kurzor után (*paste*)
P - (nagy „p” betű) vágólap tartalmának beillesztése a kurzor elé

Csere

A csere (*substitute*) általános formája **:[tartomány]s/mit/mire/[opciók] [szám]**. A „mit” egy „szabályos kifejezés”.
Az „opció” alapvetően háromféle lehet:
g - minden előfordulást lecserél (az adott sorban)
c - jóváhagy minden cserét kérdezés nélkül
i - nem különbözteti meg a kis- és nagybetűket

Néhány példa:

:s/true/false - az aktuális sorban a true első előfordulását false-ra cseréli
:s/true/false/g - az aktuális sorban a true minden előfordulását false-ra cseréli
:%s/true/false/g - a teljes fájlban a true minden előfordulását false-ra cseréli
:1,$s/true/false/g - a teljes fájlban a true minden előfordulását false-ra cseréli
:.,$s/true/false/g - a kurzor sorától a fájl végéig a true minden előfordulását false-ra cseréli
:1,.s/true/false/g - a fájl elejétől a kurzor soráig a true minden előfordulását false-ra cseréli
:s/Egyik/Másik/gi - az „Egyik” minden előfordulását „Másik”-ra cseréli ki a teljes sorban, nem kisbetű-nagybetű érzékenyen.
:1,10s/true/false/g - az első tíz sorban a true minden előfordulását false-ra cseréli
:s/true/false/g 4 - a kurzor sorától kezdve 4 sorban a „true” minden előfordulását „false”-ra cseréli

Az **&** karakter jelentése a cserében:
Az **&** karakter jelentése: „az aktuális illeszkedő szövegrész”. Ezt tehát szerepeltethetjük egyfajta „rész-stringként”, amikor azt specifikáljuk, hogy mire cserélje le az illeszkedést. Példák:
5,15s/\<fényes\>/"&"/g - az 5-15 sorok közti tartományban az összes „fényes” szót ami külön áll (tehát teljes szó és nem valami ragozott alak) idézőjelek közé rakja. Azt hogy csak a teljes szavakra keresünk rá, jelenti az, hogy a keresendő kifejezést a **\<** és **\>** jelek közé zártuk, a **&** jelenti az aktuális találatot, s mert a **&** jel idézőjelek közt szerepel, amiatt idézőjelek közé fogja zárni.
:%s/van/\U&\E/g - a „van” szót nagybetűssé alakítja mindenhol a teljes fájlban
:%s/MyString/\U&\E/g - a „MyString”-et nagybetűssé konvertálja a teljes fájlban
:%s/MyString/\L&\E/g - a „MyString”-et kisbetűssé konvertálja a teljes fájlban

Az utóbbi 3 példában a \U azt jelenti, hogy a következő karakterek a \E-ig nagybetűsek, a \L pedig azt jelenti, hogy a következő karakterek a \E-ig kisbetűsek.

:%s/MyString/&&/g - A „MyString”-et megduplázza (egymás után kétszer írja) a teljes fájlban

Az eddigi példákban nemcsak a teljes szavak cserélődtek le, hanem (ha voltak olyanok) a

szórészek is. Például ha volt egy ilyen szövegünk:
This is his idea
akkor a
:s/his/her/g
parancs hatására ezt kapjuk:
Ther is her idea
ami nyilvánvalóan nem megfelelő. Szerencsére van mód rá, hogy a Vimet rávegyük, hogy csak teljes szavakat cseréljen le(ki). Erre a fenti példánál maradva a következő parancs szolgál:
:s/\<his\>/her/
Mint látható, a lényeg az, hogy a keresendő szöveget (mit) „kacsacsőrök" közé rakjuk.
Példa a „szabályos kifejezés" használatára:
:%s/\<\(jó\|nagyszerű\)\>/remek/g
A fenti példa a teljes fájlban lecserél minden „jó" és minden „nagyszerű" szót arra, hogy „remek", DE nem cseréli le a „jó" előfordulásait szavak belsejében, azaz például a „hajó"-ból nem lesz „haremek".
:%s/\(jó\|nagyszerű\)/remek/g
Ez ugyanaz mint az előző példa, de szavak belsejében is cserél, vagyis itt a „hajó"-ból már „haremek" lesz.
:%s/red\|green\|blue/purple/g - ez a példa a teljes fájlban lecseréli a „red", a „green" és a „blue" minden előfordulását „purple"-re.
:%s/inp$/out/g – A teljes fájlban mindenütt lecseréli az „inp" stringeket „out" stringekre, de csak a sorok végén (erre utal a **$** jel)

:%s/true/false/gc - a teljes fájlban a true minden előfordulását false-ra cseréli, de minden előfordulásnál megkérdi tőlünk, hogy „komolyan gondoljuk-e". A kérdésre a következő válaszokat lehet adni:

y	- (*yes*) cserélje le ezt az illeszkedést
l	- (*last*) cserélje le ezt az illeszkedést és fejezze be a cserét
n	- (*next*) lépje át ezt az illeszkedést
Esc	- a csere befejezése
q	- (*quit*) a csere befejezése
a	- (*all*) cserélje le ezt és minden további illeszkedést
Ctrl-e	- felfelé scrollozza a képernyőt
Ctrl-y	- lefelé scrollozza a képernyőt

Továbbá, megjegyzésre érdemes, hogy a csere-parancs esetén nem okvetlenül szükséges, hogy a „/" (per) jel (angolul *slash*) legyen a határolókarakter! Ez ugyanis rendkívül kényelmetlen, ha például linuxos könyvtárnevekre keresünk rá, vagy arra akarunk egy szöveget helyettesíteni, mint amilyen például az /usr/local/. Hagyományosan ugyanis, ha a „/" karakter a szeparátor, s az „/usr/local/" stringet akarjuk a „/opt/"-tal helyettesíteni, erre a következő módon kereshetünk rá:
:s/\/usr\/local\//\/opt\//
vagyis mint látható, a „/" jeleket egy-egy „\"-el, *backlash*-sal „védtük le". No most ez rendkívül kényelmetlen, ha a kifejezésünk sok „/" jelet tartalmaz, nagy a tévesztés lehetősége. Szerencsére a Vim tud ennél jóval kellemesebb megoldást is. Az **:s** parancsot ugyanis úgy definiálták, hogy a **:s** utáni első akármilyen karakter a szeparátorjel (kivéve a **** , **"** és **|** jeleket), s ennek megfelelően – ha úgy döntünk hogy például a „#" jel lesz a

szeparátorjel – a fenti példát eként is írhatjuk:
:s#/usr/local/#/opt/#
Ez kétségkívül áttekinthetőbb!

%s/\<\(red\|green\|blue\)\>/"&"/g - e példa a teljes fájlban a „red”, a „green” és a „blue” minden előfordulását idézőjelek közé rakja. A „**&**” jelzi ugyanis az aktuális találatot (illeszkedést).
:%s/color \<\(red\|green\|blue\)\>/colored \1/g - E példa pedig a teljes fájlban mindenütt lecseréli a „color red”, a „color green” és a „color blue” kifejezéseket arra, hogy „colored *x*”, ahol „*x*” a „red”, „green”, „blue” szavak valamelyike. (A „**\1** ” jelzi a kifejezésben az első szócsoportot).

Interaktíven is cserélhetünk. Ez azt jelenti, hogy ha egy szövegben sok helyen akarjuk kicserélni ugyanazt a szót, de nem mindegyik helyen, akkor minden találatnál meg kell mondanunk a Vimnek, hogy cseréljen-e. Ennek menetét megvilágíthatjuk az alábbi példával:
Tegyük fel, hogy korábban egy vallásos témájú levelet küldtünk Máriának, s most ugyanezt el akarjuk küldeni Ildikónak is. A levélben több helyen szerepel Mária neve, ezeket kell átírni Ildikóra. De nem minden Máriát kell átírni mégsem, mert ha úgy teszünk, akkor Szűz Máriából is Szűz Ildikó lesz, ami esetleg félreértésekre adna alkalmat! Cselekedjünk ellenben a következőképpen:
Adjuk ki a
/Mária parancsot. Majd üssük be:
cwIldikó
Ez lecseréli a Mária első előfordulását Ildikóra. Most nyomjuk meg az **Esc** gombot, hogy kikerüljünk az inzert módból, s nyomjuk meg az **n** gombot, hogy a következő találatra ugorjunk. Ha ezt is le akarjuk cserélni, elég ha megnyomjuk a „**.**” (pont) gombot, s ez megismétli az előző csere parancsot. Ha nem akarjuk e találatot lecserélni, megint **n**-et kell nyomnunk. S így az **n** és a pont nyomogatásával haladhatunk végig a dokumentum összes találatán.

Ha csak ki akarunk törülni egy szót a dokumentumban mindenütt, de semmit sem kívánunk a helyére írni, akkor használjuk e formát:
:%s/TörlendőSzó//g

Könyvjelzők

26 darabot lehet letenni, a-z-ig minden betű használható. Kétféleképpen lehet ugrani egy könyvjelzőhöz, a ` (backtick) vagy a **'** (aposztróf) paranccsal. A backtick pont a lerakott markerre ugrik, az aposztróf a marker sorának első, nem whitespace karakterére. A könyvjelzők ugyanúgy használhatók pl. másoláshoz vagy törléshez, mint más mozgásparancsok, ha sok sort kell másolni, egyszerűbb, mint számolgatni. Példák:

ma - lerakja az a markert
`a - ugrás „a”-hoz
'a - ugrás „a” sorába
y`f - másolás a kurzortól az „f” markerig
d'd - törlés a „d” marker soráig
:'a,'bs/ez/az/g - az „a” és a „b” könyvjelzők közötti sorokban kicseréli az „ez” minden előfordulását „az”-ra
:marks - a könyvjelzők listáját adja vissza
:delm a - Törli az „a” könyvjelzőt
:delm a b c - Törli az a, b és c könyvjelzőket
:delm abc - Törli az a, b és c könyvjelzőket
:delm p-z - Törli az összes könyvjelzőt p -től z-ig
:delm vagy **:delmarks** - Törli az adott buffer összes könyvjelzőjét

Ha a könyvjelzők nagybetűs változatait használjuk, (például **mA**) akkor azok alkalmazhatók fájlok között is.
Ha egy nagybetűs könyvjelzőt hozol létre mondjuk „A”-t az **a.txt** nevű fájlban és utána ugyanabban az ablakban megnyitod a **b.txt**-t majd kiadod az **'A** parancsot, akkor a Vim visszaugrik az **a.txt** arra a sorára, ahol „A”-t definiáltad. (feltéve hogy nem módosítottad a **b.txt** -t vagy be van kapcsolva a *hidden config* valtozó).

Meg kell említeni még a
`` parancsot is. Ezzel visszaugorhatunk oda, ahonnan korábban elugrottunk. Tehát ha mondjuk elugrottunk a harmadik sor elejéről az m-nevű könyvjelzőre a **'m** paranccsal, majd ott csinálunk valamit, például kijelöljük a szöveget és végrehajtunk rajta egy parancsot, akkor ezután a `` parancs hatására visszaugrunk oda, ahonnét az **'m** -re ugrottunk, tehát a harmadik sor elejére.
A **''** parancs (két aposztróf) ugyanaz, mint a `` .

Bufferek

Az alapfogalmak:
–A buffer, az egy fájl memóriában levő része.
–A képernyő, az a buffer látható része.
–A Tab, az a képernyők egy gyűjteménye.

Az ablak, az tehát egyfajta „rátekintés” a bufferre. Használhatsz több ablakot is ugyanarra a bufferre.
A buffer, az egy fájl amit betöltöttél a memóriába szerkesztésre. Az eredeti fájl változatlan marad, amíg a módosított változatot el nem mented.

Egy buffer 3 állapotban lehet:

–**Aktív**: A buffer meg van jelenítve egy képernyőn.
–**Rejtett**: A buffer nincs megjelenítve.
–**Inaktív**: A buffer nincs megjelenítve, és nem tartalmaz semmit.

A bufferekre vonatkozó parancsok:

:ls	Kilistázza a buffereket
:files	Kilistázza a buffereket
:buffers	Kilistázza a buffereket
:b2	A 2-es bufferre ugrik (azaz azt jeleníti meg az aktuális ablakban)
:q	Bezárja az aktuális buffert (a listában megmarad)
:bd	Bezárja és a listából is törli az aktuális buffert
:sb 2	Kettéosztja a képernyőt, s felül megjeleníti a 2-es buffer tartalmát
:bn	Ugrás a következő bufferre
:bp	Ugrás az előző bufferre
:bf	Ugrás az első bufferre
:bl	Ugrás az utolsó bufferre
:buffer *N*	Megnyitja az *N*-edik buffert
:b *N*	Megnyitja az *N*-edik buffert
:sbn *N*	Kettéosztja a képenyőt, és megnyitja az *N*-edik buffert
:sbn	Kettéosztja a képenyőt, és megnyitja a következő buffert
:sbp	Kettéosztja a képenyőt, és megnyitja az előző buffert

A bufferek közti ciklikus lépkedésre csinálhatunk saját billentyűparancsokat is. Ehhez vegyük fel a következő két sort a **.vimrc** fájlba:
nnoremap <F3> :bnext<CR>
nnoremap <F2> :bprevious<CR>
Ezután (miután újraindítottuk a Vimet) az **F2** billentyű az előző, az **F3** pedig a következő bufferre ugrik.

Regiszterek

Regiszterekből 9 fajta létezik a Vimben:

1.A névtelen regiszter **""**
2.Tíz számozott regiszter **"0** -tól **"9** -ig
3.A törlés-regiszter (*delete register*) **"-**
4.26 db, betűvel jelölt regiszter **"a** -tól **"z** -ig vagy **"A** -tól **"Z** -ig
5.Négy „csak olvasható" (*read-only*) regiszter: **":**, **".**, **"%** és **"#**
6.Az expression regiszter: **"=**
7.A szelekciós regiszterek: **"*** , **"+** és **"~**
8.A „feketelyuk" regiszter (*black hole register*): **"_**
9.Az utolsó keresés regisztere (*last search pattern register*): **"/**

1.A névtelen regiszter: ""
A Vim e regisztert a „d", „c", „s", „x", „y" parancsok használata esetén tölti fel értékkel, amennyiben a parancshoz nem adtunk meg specifikusan valamely másik regisztert (pld **"xdd**).

Továbbá, a Vim felhasználja e regiszter tartalmát a „p" és a „P" parancsok esetén is, ha nem adunk meg hozzájuk valamely más regisztert. E regiszterre közvetlen módon hivatkozhatunk a **""** névvel (két idézőjel).

2.A számozott regiszterek: "0 -tól "9 -ig
A 0-ás regiszter a vágólap, ezen dolgoznak a dd, yy, p parancsok. Az 1-9 regiszterek mindig az utolsó 9 törölt szöveget tartalmazzák, ha törlünk valamit, az az 1-esbe kerül, a 9-es elveszik, az 1-es a 2-esbe másolódik, és így tovább. Az 1-es ugyanaz, mint a 0-ás, de lehet fájlok között is használni.

3.A törlés-regiszter (delete register): "-
Ezt a regisztert akkor használja a Vim, ha kevesebb mint egy sort törlünk – kivéve ha a parancs konkrétan tartalmazza, melyik regisztert kell használnia.

4.A betűvel jelölt regiszterek "a -tól **"z** -ig vagy **"A** -tól **"Z** -ig
A Vim e regisztereket csak akkor használja, ha „azt mondjuk neki". Ha a kisbetűs elnevezésüket használjuk, akkor lecseréli a korábbi tartalmukat az új eredményre, ha nagybetűsként használjuk az elnevezésüket, akkor az új tartalmat hozzáfűzi az előző (addigi) tartalmukhoz.

5.A csak olvasható regiszterek (*Read-only registers***): ":**, **".**, **"%** és **"#**
Ezek a '**%** ', '**#** ', ':' és '.'. Ezeket csak a „p", „P", és a „:put" parancsoknál használhatod, illetve a CTRL-R esetében.

A **".** tartalmazza az utolsó beszúrt szöveget.
A **"%** tartalmazza az aktuális fájl nevét.
A **"#** tartalmazza a megváltozott fájl nevét.
A **":** tartalmazza az utoljára végrehajtott parancssort.

6.Az expression regiszter: "=
Ez nem egy valódi regiszter, mely szöveget tartalmaz, hanem egy módszer kifejezések használatára egy olyan parancsban, mely egy regisztert használ. Az expression regiszter csak olvasható, nem lehet bele szöveget másolni.

7.A szelekciós regiszterek: "*, "+ és **"~**
E regisztereket használhatod a GUI által is használt vágólapok számára szövegek tárolására. Ha a vágólap nem elérhető vagy nem működik, helyettük a *névtelen regiszter* lesz használva. Az MS-Windows alatt a **"*** és az **"+** regiszterek ugyanazok (egybeesnek). Lásd még e könyvben a „**Vágólapra másolás**” című fejezetet.

A csak olvasható **"~** regiszter tárolja az utolsó „fogd és vidd” (*drag'n'drop*) művelet szövegét.

8.A feketelyuk-regiszter: "_
Ha ebbe a regiszterbe írunk, semmi sem történik. Ezt akkor használjuk, ha úgy akarunk törölni egy szöveget, hogy nem bántjuk a normál regiszterek tartalmát. Ha olvassuk e regiszter tartalmát, semmit sem kapunk vissza eredményül.

9.Az utolsó keresés regisztere (*last search pattern register*): **"/**
Az utolsó keresés mintáját tartalmazza. Ezt használja az „n” és a 'hlsearch'. Írható a „:let” paranccsal, de nem lehet bele másolni („y”), s a törlési parancsok sem másolnak bele szöveget.

Egy példa e regiszter írására: **:let @/ = "the"**
Ha valamely adatkiviteli (*put*) parancsot használunk regiszter közvetlen megadása nélkül, a Vim az utoljára feltöltött regisztert használja, mely általában a névtelen regiszter. Ha tudni akarod, melyik regiszter mit tartalmaz, használd a
:dis
parancsot. Ekkor a Vim kiírja a parancssor fölé egy kis ablakba az összes regiszter tartalmát, a névtelen regiszter címkéje **""** lesz.

A regiszterekre vonatkozó parancsok közül néhány fontosabb:

"aY - az „a” regiszterbe másolja az aktuális sort
"AY - hozzáadja az „a” regiszterhez az aktuális sort
"ap - a-t bemásolja a kurzor után
"b10yy - b-be másolunk a kurzor sorától kezdve 10 sort
"cy`f - másolás a „c” regiszterbe a kurzortól az „f” markerig
"cP - „c” tartalmának beszúrása a kurzor elé
"cp - „c” tartalmának beszúrása a kurzor mögé

A 10, számmal azonosított regiszterre ugyanúgy idézőjellel lehet hivatkozni, mint a betűsekre.

"0p - ugyanaz, mint **p**

Műveletek regiszterekkel

:let @a=@_ - az „a” regiszter törlése
:let @a="" - az „a” regiszter törlése
:let @a=@" - a „névtelen regiszter” elmentése az „a” regiszterbe
:let @*=@a - az „a” regiszter másolása a vágólapra
:let @+=@a - az „a” regiszter másolása a vágólapra
:let @*=@: - az utolsó parancs másolása a vágólapra
:let @*=@/ - az utolsó keresés másolása a vágólapra
:let @*=@% - az aktuális fájlnév másolása a vágólapra

A vágólapról részletesebben lásd a következő fejezetet.

Vágólapra másolás[1]

Ahhoz, hogy Vim-ben kimásolt szöveget külső alkalmazásokban is elérjük, a Vim e célra fenntartott regisztereibe kell másolni, mely megyezik azzal, mintha pl. egy böngészőből CTRL+C-vel (+) vagy kijelöléssel (*) másoltunk volna valamit a vágólapra. Ilyenkor a szokásos másolási műveletet egy `"+` vagy `"*` kombinációnak kell megelőznie. Néhány példa.:

"*yw - másolás a kurzortól a szó végéig
"+dd - az aktuális sort törli és másolja vágólapra
"*yy - az aktuális sort másolja vágólapra
ggVG"+y - az egész dokumentumot kijelöli, majd azt másolja a vágólapra.

Meg kell jegyezni, hogy tulajdonképpen nem is csupán egy, hanem 3 vágólap létezik a Vim számára! Az egyik a maga szokásos, Vim-beli műveletei számára – ezt nem használhatjuk külső alkalmazások számára. A másik kettő pedig a **"*** és a **"+** regiszterek, amelyeket már használhatunk arra, hogy – például – a LibreOffice-ba beillesszük a tartalmukat, ám e két regiszter nem egyforma! Ha ugyanis a **"*** regiszter tartalmát akarjuk beilleszteni valahová, akkor erre a **középső egérgombot** kell használnunk, ha ellenben a **"+** regisztert használjuk, akkor annak tartalmát a **Ctrl-v vagy a Shift-Insert billentyűkombinációval** illeszthetjük be. Tehát:

"+ - **Ctrl-v** vagy **Shift-Insert**
"* - **középső egérgomb**

Ahhoz hogy a Vimbe illeszthessünk más alkalmazásból szöveget, azt másoljuk a vágólapra (például a böngészőből vagy a LibreOffice-ből egérrel kijelölve majd Ctrl-C), aztán a Vimben használjuk beillesztésre a középső egérgombot, vagy adjuk ki parancsmódban (normál módban) a következő parancsot:
"+p

1 E fejezethez felhasználva a HUP wiki anyaga is: http://wiki.hup.hu

Vizuális mód

A vizuális mód lehetőséget ad arra, hogy jól láthatóan kiválasszuk azt a szövegrészt, amin a következő műveletet akarjuk végrehajtani. Nincs szükség a sorok számolgatására. A vizuális módot a következőképpen használjuk:

1. bekapcsoljuk a vizuális módot, ezzel egyben az aktuális kurzorpozíció helyén ki is jelöltük a szövegblokk egyik határát.
2. bármilyen mozgásparanccsal kijelöljük a blokk másik határát
3. a kijelölt szöveggel csinálunk valamit, másolunk, kivágunk, stb.

Vizuális mód tulajdonképpen háromféle van, a bekapcsolás módjától függően, az aktuális módot a státuszsor jelzi:

v normál mód (VISUAL), kijelölés karakterenként

Shift-v sor mód (VISUAL LINE), kijelölés soronként

Ctrl-v blokk mód (VISUAL BLOCK), négyszögletes terület jelölhető ki

A kijelölés után leggyakrabban az x (kivágás) és y (másolás) parancsokat használjuk. Néhány érdekesebb vizuális módbéli parancs még:

> - jobbra tolja a kijelölt szövegrészt
< **-** balra tolja a kijelölt szövegrészt
~ - megfordítja a kijelölést, kisbetűből nagyot csinál és fordítva
u **-** a kijelölést kisbetűsre konvertálja
U - a kijelölést nagybetűsre konvertálja
A - ha blokk módban vagyunk és a blokk végét a $ paranccsal (sor vége) jelöltük ki, akkor a blokk minden sorának a végére odaírja a begépelt szöveget

Azt is megtehetjük, hogy miután „vizuális üzemmódba" kapcsoltuk a Vimet, itt a kurzormozgató billentyűk és parancsok valamelyikével kijelölünk egy szövegrészt (ami inverzbe kerül), majd megnyomjuk a „**:**" gombot (kettőspontot) hogy a parancssorba kerüljünk. Oda erre automatikusan beíródik a

:'<,'>

karaktersorozat, jelezve ezzel, hogy az ezutáni parancs az imént kijelölt szövegrészre fog vonatkozni. Ezután már csak a parancsot kell begépelnünk, s örülhetünk. A parancs ilyenkor általában a csere-parancs szokott lenni, vagyis az ilyesféleképp néz majd ki:

:'<,'>s/mit/mire/g

gv - ez a parancs újra kiválasztja (kijelöli) a legutoljára vizuális módban kijelölt területet.

A részleteknek utánanézhetsz a **:help visual_mode** paranccsal.

Egyéb hasznos parancsok

. (pont) megismétli az utolsó parancsot
:!parancs kilép a shellbe és végrehajtja a parancsot (például !ls)
:r !parancs kilép a shellbe és végrehajtja a parancsot, s a parancs eredményét beilleszti a kurzor aktuális helyére (például :r !ls)
:0r !parancs kilép a shellbe és végrehajtja a parancsot, s a parancs eredményét beilleszti az aktuális buffer első sorától kezdve (például :0r !ls)
:.!parancs kilép a shellbe és végrehajtja a parancsot, s a parancs eredményét beilleszti a kurzor aktuális helyére
& Megismétli az utolsó csere parancsot
; Megismétli az utolsó f, F, t vagy T parancsot
, Az utolsó f, F, t vagy T parancs, de visszafelé
Ctrl-g Kiírja alulra a fájl nevét, hogy módosított-e, s hogy hány sorból áll.
:sort A kijelölt sorokat ABC sorrendbe rendezi
:sort i A kijelölt sorokat ABC sorrendbe rendezi, de nem kisbetű-nagybetű érzékenyen
:sort! A kijelölt sorokat ABC szerinti CSÖKKENŐ sorrendbe rendezi
:sort! i A kijelölt sorokat ABC szerinti CSÖKKENŐ sorrendbe rendezi, nem kisbetű-nagybetű érzékenyen
:sort u Úgy rendez, hogy ha többször is szerepel ugyanaz a sor, akkor csak az egyiket hagyja meg (duplikátumok kiszűrése)
:sort! u Mint :sort u, de csökkenő sorrendbe rendez
!!date Beszúrja az aktuális dátumot és időt a kurzor pozíciójához
ga Alul a státuszsorban megmutatja a kurzor alatti karakter Ascii kódját decimálisan, hexában és oktálisan. A „ga” a „**g**et **a**scii” rövidítése.
:as Ugyanaz mint a **ga** parancs
:ascii Ugyanaz mint a **:as**
:!parancs % Kilép a shellbe, és végrehajtja a parancsot, úgy, hogy argumentumként átadja neki a Vim aktuális fájlját. Példa: **:!wc %**
:<Ctrl-r>" - vagyis ha a „/” után megnyomjuk a Ctrl-r és az " (idézőjel) billentyűket, akkor a parancssorba behúzza az utoljára másolt szöveget.

:e $MYVIMRC a $HOME/.vimrc file megnyitása szerkesztésre
:echo változó a Vim valamely változójának kiiratása. Példák:
:echo $MYVIMRC
:echo $HOME
:echo $VIM

Ctrl-a Inkrementálja (eggyel megnöveli) azt a számot, amin a kurzor épp áll. Ha a kurzor nem számon áll, akkor a sorban található következő számot növeli meg eggyel.
Ctrl-x Dekrementálja (eggyel csökkenti) azt a számot, amin a kurzor épp áll. Ha a kurzor nem számon áll, akkor a sorban található következő számot csökkenti eggyel.
A **Ctrl-a** és a **Ctrl-x** parancsok nemcsak decimális számokra működnek, hanem hexadecimális és oktális számokra is. A hexadecimális számok a **0x** az oktális számok a **0** (nulla) karakterrel kell kezdődjenek.
g majd **Ctrl-g** Kiírja a státuszsorba a file-ben (az aktuális bufferben) levő szavak

és bájtok számát, meg azt is, hogy a kurzor hol áll éppen.

g~ A (vizuális módban kiválasztott) szövegben átváltja a betűket: kisbetűből nagybetű lesz és fordítva.

gU A (vizuális módban kiválasztott) szövegben a betűket nagybetűssé konvertálja.

gu A (vizuális módban kiválasztott) szövegben a betűket kisbetűssé konvertálja.

:%!xxd A hexa editor bekapcsolása

:%!xxd -r A hexa editor kikapcsolása

:version Kiír egy csomó információt a Vim aktuális verziójáról. Az enyém például ezt:

```
VIM - Vi IMproved 7.3 (2010 Aug 15, compiled May 21 2011 13:40:12)
Compiled by pv@Csiszilla
Normal version with GTK2 GUI.  Features included (+) or not (-):
-arabic +autocmd +balloon_eval +browse +builtin_terms +byte_offset +cindent +clientserver +clipboard +cmdline_compl
+cmdline_hist +cmdline_info +comments -conceal +cryptv -cscope +cursorbind +cursorshape +dialog_con_gui +diff +digraphs +dnd
 -ebcdic -emacs_tags +eval +ex_extra +extra_search -farsi +file_in_path +find_in_path +float +folding -footer +fork()
+gettext -hangul_input +iconv +insert_expand +jumplist -keymap -langmap +libcall +linebreak +lispindent +listcmds +localmap
-lua +menu +mksession +modify_fname +mouse +mouseshape -mouse_dec +mouse_gpm -mouse_jsbterm -mouse_netterm
-mouse_sysmouse
+mouse_xterm +multi_byte +multi_lang -mzscheme +netbeans_intg -osfiletype +path_extra +perl +persistent_undo +postscript
+printer -profile +python -python3 +quickfix +reltime -rightleft -ruby +scrollbind +signs +smartindent -sniff +startuptime
+statusline -sun_workshop +syntax +tag_binary +tag_old_static -tag_any_white -tcl +terminfo +termresponse +textobjects
+title +toolbar +user_commands +vertsplit +virtualedit +visual +visualextra +viminfo +vreplace +wildignore +wildmenu
+windows +writebackup +X11 -xfontset +xim +xsmp_interact +xterm_clipboard -xterm_save
   system vimrc file: "$VIM/vimrc"
     user vimrc file: "$HOME/.vimrc"
      user exrc file: "$HOME/.exrc"
  system gvimrc file: "$VIM/gvimrc"
    user gvimrc file: "$HOME/.gvimrc"
    system menu file: "$VIMRUNTIME/menu.vim"
  fall-back for $VIM: "/Programs/Vim/7.3/share/vim"
Compilation: gcc -c -I. -Iproto -DHAVE_CONFIG_H -DFEAT_GUI_GTK  -pthread -I/usr/include/gtk-2.0 -I/usr/lib/gtk-2.0/include -I
/Programs/ATK/1.17.0/include/atk-1.0 -I/usr/include/cairo -I/usr/include/pango-1.0 -I/usr//include/glib-2.0 -I/usr//lib/glib-
2.0/include -I/usr/include/freetype2 -I/Programs/LibPNG/1.2.16/include/libpng12 -I/usr/include/pixman-1     -O2 -march=i686 -
fomit-frame-pointer -pipe -D_FORTIFY_SOURCE=1  -I/Programs/Xorg/7.2/include
Linking: gcc -L/Programs/ATK/1.17.0/lib -L/usr//lib   -L/Programs/Xorg/7.2/lib   -Wl,-E   -L/usr/local/lib -o vim   -pthread
-L/Programs/ATK/1.17.0/lib -L/usr//lib -lgtk-x11-2.0 -lgdk-x11-2.0 -latk-1.0 -lgdk_pixbuf-2.0 -lpangocairo-1.0 -lpango-1.0 -l
cairo -lgobject-2.0 -lgmodule-2.0 -lgthread-2.0 -lrt -lglib-2.0   -lXt -lncurses -lgpm -Wl,-E  /System/Links/Libraries/perl5/
5.8.8/i686-linux/auto/DynaLoader/DynaLoader.a -L/Programs/Perl/5.8.8/lib/perl5/5.8.8/i686-linux/CORE -lperl -lcrypt -lutil -l
c -L/Programs/Python/2.5.2/lib/python2.5/config -lpython2.5 -lutil -lm -Xlinker -export-dynamic
```

Ablakok

:split	az aktuális ablakot vízszintesen kettéosztja
:vsplit	az aktuális ablakot függőlegesen kettéosztja
Ctrl-w nyíl	a nyíl szerinti irányban levő ablakba ugrik
Ctrl-w +	egy sorral megnöveli az aktuális ablak méretét
Ctrl-w -	egy sorral csökkenti az aktuális ablak méretét
Ctrl-w 4+	négy sorral megnöveli az aktuális ablak méretét
Ctrl-w 4-	négy sorral csökkenti az aktuális ablak méretét
Ctrl-w >	egy oszloppal növeli az aktuális ablak méretét
Ctrl-w <	egy oszloppal csökkenti az aktuális ablak méretét

Az ablakok elhelyezkedésének megváltoztatása:
Tegyük fel, hogy van kettő vagy több ablakod egymás felett, tehát a képernyőd vízszintesen van ketté (vagy többfelé) osztva, de te függőleges felosztást szeretnél. Ehhez add ki a
:windo wincmd H
parancsot. A visszatérés a vízszintes felosztáshoz a
:windo wincmd K
paranccsal lehetséges.

Egy hasznos trükk: Vegyük fel az alábbi két sort a **.vimrc** fájlba:
map <C-Up> <C-W>k
map <C-Down> <C-W>j

Ennek hatására ha az aktuális ablak fölötti ablakra akarunk ugrani, azt megtehetjük a **Ctrl-Kurzorfel** gombbal, hasonlóképp az aktuális ablak alatti ablakra ugorhatunk a **Ctrl-Kurzorle** gombbal.

Undo és redo

u - *undo* - utolsó változtatás megszüntetése, ha mégegyszer megnyomod, akkor meglepő módon *redo*-t (undo undo) csinál

. - *redo* - az utolsó módosítást ismétli. u után redo undo, azaz ha egy lépésnél többet akarsz undo-zni, akkor a pontot kell nyomkodni

U - undo - az aktuális sorban történt összes változtatás visszacsinálása, ez csak addig megy, amíg ki nem megyünk a sorból

Ctrl-r - ismét (a visszavonás ellentettje)

Makrók[2]

A :map-pel egy billentyűhöz rendelhetünk tetszőleges parancsokat (a kettőspontot és az u-t nem lehet átdefiniálni). Adott például a következő kódrészlet:

```
m_action = action;
m_subsystem = subsystem;
m_eventTrigger = eventTrigger;
m_subEventTrigger = subEventTrigger;
```

Tegyük fel, hogy éppen a fordított értékadásokra lenne szükségünk. Ezért definiáljunk a q-ra egy makrót, ami felcseréli nekünk a két azonosítót:

:map q ^2x2wim_^[buta, egyszerű

:map q "adedf f;Pt;"ap intelligens, bonyolult

Figyeljük meg, hogy az első változatban az inzert módból való kilépéshez szükséges Esc-t a **^[** karaktersorozat jelzi. Ezt és a többi vezérlőkaraktert úgy lehet előállítani, hogy a megfelelő billentyű (esetünkben az Esc) lenyomása előtt egy **Ctrl-v**-t nyomunk.

Billentyűzet átdefiniálás

A **:map** kiválóan alkalmazható a nyilak, PgUp, PgDown, Home, End gombok életrelehelésére is, ha esetleg nem működnének rögtön induláskor is. (Általában működni szoktak). Pld:

:map ^[[6~ ^F a PgDown-hoz rendeli a Ctrl-f-et.

A **^[[6~** hieroglifa a **Ctrl-v** majd a **PgDown** lenyomásával állítható elő, a **^F** egyszerűen egy **Ctrl-f**. Ezeket be lehet tenni az ~/.exrc-be is, az ebben leírt parancsokat a Vim induláskor lefuttatja. Pld. a

```
map ^[[6~ ^F
map ^[[5~ ^B
map ^[OD h
map ^[OC l
map ^[OA k
map ^[OB j
```

tartalmú .exrc fájl definiálja nekünk a négy nyilat és a PgUp/PgDown gombokat, de ez gépfüggő. A kontrolszekvenciákat persze itt is a Ctrl-v-vel kell beadogatni.

Makrókat nemcsak a **:map** paranccsal vehetünk fel, hanem „menet közben” is. Van rá mód ugyanis, hogy amíg le nem állítjuk, a Vim minden gombnyomásunkat rögzítse.

qa - felvétel indítása az „a” bufferbe (az állapotsorban „recording” látható)
q - felvétel befejezése
@a - az „a” buffer tartalmát parancsként értelmezi
@@ - az utoljára végrehajtott makró futtatása

Ne feledjük, hogy ezek ugyanazok az a-z bufferek, amiket a másolásnál használunk, tehát

2 E fejezet megírásához felhasználtam *Vajkó Péter* internetre feltöltött anyagát is.

pl. egy **qf** parancs felül fogja írni az f bufferünket.

Egy példa a fentiekre: sorszámok generálása makróval:

Üssünk **Esc**-t hogy parancs módba kerüljünk, majd **i**-t hogy inzert módban legyünk. Ezután számozzuk meg 1-el az első sort, azaz üssük be:
1.
Esc-vel lépjünk ki az inzert módból, majd üssük be:
qa
A „q" indítja a makró felvételét, mely most az „a" bufferbe fog történni.
Üssük be: **Esc yy p**
Az **Esc** azért kell, hogy a makró parancsmódban induljon, az **yy** kimásolja a sort, a **p** pedig beilleszti.
Most nyomjuk le a **Ctrl-a** billentyűkombinációt, mely inkrementálja a sorszámot.
Ezután üssük le a „**q**" gombot, ezzel befejezve a makró felvételét.
Majd futtassuk le 98-szor a fenti makrót, a **98@a** paranccsal.
Az eredmény 100 db, megsorszámozott sor lesz.

Színsémák

A forráskódok szövegét a Vim színezi. Többféle színsémája van, amiből válogathatunk. Sajnos az alapértelmezett (a „default”) nem valami jó, mert a megjegyzések színét sötétkékkel színezi, ami a fekete alapon (szerintem) olvashatatlan. Ezért ajánlatos áttérni egy másik színsémára – én a „desert” nevűt találtam jónak. Az alábbi nevűekből válogathatunk:

blue.vim	shine.vim	murphy.vim
default.vim	torte.vim	peachpuff.vim
desert.vim	darkblue.vim	ron.vim
evening.vim	delek.vim	slate.vim
morning.vim	elflord.vim	zellner.vim
pablo.vim	koehler.vim	

De egyéb színsémákat is letölthetünk a
http://www.vim.org/scripts/script_search_results.php?script_type=color+scheme
oldalról. Ahhoz hogy a letöltött színsémát használhassuk, azt be kell másolni a megfelelő könyvtárba, mely például a GoboLinux disztribúció alatt a /Programs/Vim/Current/Shared/vim/vim73/colors

Adott színsémára a „colorscheme” paranccsal lehet váltani, az alábbi példa szerint:

:colorscheme desert

Ha ezt beírjuk a $HOME/.vimrc fájlunkba, akkor már induláskor eszerint viselkedik.
Ha paraméterek nélkül adjuk ki a **:colorscheme** parancsot, akkor alulra a parancssorba kiírja az aktuálisan használt színséma nevét, például azt, hogy „desert”.

Beállítások

A **:set** paranccsal állíthatjuk be a Vim működését befolyásoló mindenféle opciókat. A kétállapotú opciók beállítása a **:set valami**, törlése a **:set novalami** paranccsal történik. Lássuk a leghasznosabbakat:

:set all - megmutatja az aktuális beállításokat
:set showmode - a jobb alsó sarokba mindig kiírja, ha inzert vagy átírási módban vagyunk (CHANGE/INSERT MODE)
:set noautoindent - kikapcsolja az autoindentet. Baromi hasznos, amikor xterm-ek között a középső egérgombbal másolunk, nem lesz lépcsőzetesen eltolva a bemásolt szöveg
:set noai - ugyanaz mint a **:set noautoindent**
:set number - beszámozza a sorokat
:set ic - kereséskor nem különbözteti meg a kis és nagybetűket (*ignorecase*)
set incsearch - inkrementális keresés, miközben gépelem a keresendő szöveget, már keres, nem vár ENTER-re
set ruler - mutatja a sor- és oszlopkoordinátákat

Persze az állandóan használt opciókat is be lehet tenni az .exrc-be.

A help parancs

A Vim rendelkezik egy kiváló (angol nyelvű) súgóval is, amit a **:help** vagy röviden a **:h** paranccsal érhetünk el. Ekkor a Vim kettéosztja a képernyőt, s a súgó szövege a felső részben érhető el. Rákereshetünk közvetlenül egy-egy parancsra is, a **:help parancs** által, például az **:s** (a csere) parancsra a **:help :s** módon.
A súgóablakból a **:q** megnyomásával léphetünk ki.

Ha egy bizonyos parancsra akarunk rákeresni, de nem vagyunk biztosak benne, hogy az pontosan miként is szerepel a súgóban, akkor tegyük azt, hogy a :help után beírjuk a parancs egy részletét, például az elejét, majd nyomjuk meg a
Ctrl-d billentyűkombinációt. Ekkor a Vim kilistázza az összes olyan kulcsszót, amiben szerepel a begépelt string. Példa: **:help com<Ctrl-d>** (A „<Ctrl-d>" nem azt jelenti, hogy így kell beütni karakterenként, hanem hogy egy gombnyomásra kell a Ctrl-d billentyűkombinációt lenyomni!)

Ha a súgó ablakában vagyunk, abban ugyanúgy mozoghatunk, mint bármi más „csak olvasható" (readonly) dokumentumban. Viszont rendelkezésünkre áll egy hasznos billentyűkombináció:
Ctrl-] - Ez, ha egy kulcsszón állunk, elugrik annak a kulcsszónak a súgófejezetéhez. A kulcsszavak a helpben kék színnel vannak jelezve. Megjegyzendő, hogy magyar kiosztású billentyűzeten a „**]**" az **AltGr-g** billentyűkombinációval érhető csak el, emiatt a **Ctrl-]** helyett ekkor tulajdonképpen a **Ctrl-AltGr-g** billentyűkombináció a használandó!

Az előző keresések és parancsok története

Amikor lenyomjuk a „*/* ” gombot a keresés megkezdéséhez, akkor a Vim kiírja alul a parancssorba is a „*/* ” jelet. Ha ekkor megnyomjuk a kurzorbillentyűk közül a felfelenyilat, bemásolja oda az előző keresés parancsát. A kurzorfel-nyíllal mindig egy előző keresés-parancsra ugorhatunk, a kurzorle-nyíllal az előzőre. Az Enterrel aktivizálhatjuk a kívánt parancsot. Hosszú parancsoknál ez nagyon hasznos, mert nem kell a bonyolult parancsot újra begépelni. Ugyanez érvényes a visszafelé keresésre, a „**?**” paranccsal.

Hasonlóképp, ha *bármi* parancsot is akarunk kiadni, melyet már használtunk egyszer, akkor csak üssük le a kettőspontot (melyet a Vim kiír a parancssor elejére), majd használjuk a fenti módon a kurzornyilakat a parancsok listázásához.

Van azonban más módszer is. A fenti módszerrel ugyanis egyszerre mindig csak az egyik parancsot (vagy keresést) láthatjuk. Üssük azonban be:
q/ vagy **q?**
Erre alul a parancssor fölött megjelenik egy kis, néhány soros ablak, amelynek minden sora az előző keresések egy-egy parancsát tartalmazza. Ezen ablakban a kurzornyilakkal mozoghatunk fel-le (az sem baj ha nem fér ki az összes keresési parancs az ablakba, mert a legfelső sorában kurzorfel-nyilat nyomva az scrollozódni fog), sőt, az sem baj ha nincs olyan parancs, mely teljesen megfelelne az elképzeléseinknek: a parancsokat ugyanis a Vim szokásos eszközeivel nyugodtan szerkeszthetjük is, s így válasszuk ki a szükségeshez legjobban hasonlító parancsot, szerkesszük át, majd nyomjunk Entert, s máris örülhetünk, mert megmenekedtünk egy rakás gépeléstől!

Másik módszer e keresési lista előhívására, hogy lenyomjuk a */* gombot, mire azt ki is írja a parancssor elejére, majd megnyomjuk a **Ctrl-f** billentyűkombinációt.

Hasonlóképp, a
q:
parancs a korábban kiadott parancsokat jeleníti meg egy kis ablakban, melyet az előzőekben, a **q/** parancsnál leírt módon használhatunk.

Másik módszer e parancslista előhívására, hogy lenyomjuk a **:** gombot, mire azt ki is írja a parancssor elejére, majd megnyomjuk a **Ctrl-f** billentyűkombinációt.

Akár a keresés-, akár a parancstörténet kis ablakában állunk épp, ha meggondoltuk magunkat és nem kívánjuk azt épp használni, a **Ctrl-c** kétszeri megnyomásával kiléphetünk belőle (és a parancssorból).

Másik módszer a parancs- vagy keresésitörténet-ablak bezárására, hogy megnyomjuk a **:** (kettőspont) gombot, erre a parancssorba jutunk, majd ezután **q**-t nyomunk (és Entert).

Ha a */* után megnyomunk egy tetszőleges billentyűt, például /a akkor utána a kurzornyilak mozgatásával csak az azzal a betűvel kezdődő kereséseket listázza ki a parancssorba a Vim. Sőt, ez nemcsak egyetlen karakterre érvényes, hanem tetszőleges hosszúságú keresési kezdetre is: csak a begépelt stringgel kezdődő kereséseket listázza.

A fentiekhez hasonlóan, ha a **:** (kettőspont) után begépeljük bármi parancs elejét, utána a kurzornyilak mozgatására csak az azzal kezdődő parancsok közül válogathatunk.

A **:his** illetve a **:history** parancs kilistázza a képernyőre az összes korábban használt parancsot, a **:his /** parancs pedig az összes korábbi keresést listázza ki.

A **@:** regiszter tartalmazza a legutolsó kiadott parancsot, hasonlóképp a **@/** tartalmazza az utoljára végrehajtott keresést. Ennek megfelelően a
@:
parancs kiadása mindig az utolsó kiadott parancsot ismétli meg. Valamint, a
":p
parancs beilleszti nekünk a szövegbe az utolsó végrehajtott parancsot, a
"/p
parancs pedig beilleszti az utolsó végrehajtott keresést.

A tab-ok használata

A tabok lehetővé teszik több fájl használatát ugyanazon a képernyőn. Olyasfélék, mint a Firefox tab-jai (fülei). A Vim egyszerre – alapértelmezett beállításként – max 10 tabot tud kezelni, de ennek értéke átállítható a **.vimrc** fájlban, a **tabpagemax** változó segítségével, az alábbi példa szerint:
set tabpagemax=15

A tabok parancsai:

:tabnew file - megnyitja a fájlt egy új tabon
:tabnew - megnyit egy üres tab-ot
gt - vált a következő tabra
gT - vált az előző tabra
:tabn - vált a következő tabra
:tabp - vált az előző tabra
:tabfirst vagy **:tabfir** vagy **:tabr** - vált az első tabra
:tabl vagy **:tablast** - vált az utolsó tabra
:tabf filenév - megnyitja egy új tabon az adott nevű állományt. Használható a filenévben a * karakter is
:tabs - felsorolja alul a parancssor fölött egy kis ablakban a tabokat, a bennük megnyitott állományok nevével
:tabc vagy **:tabclose** - bezárja az aktuális tabot
:tabm N - átmozgatja az aktuális tabot az N-edik utánra. Ha a legelsőnek akarjuk, akkor használjuk a **:tabm 0** parancsot.
:tabmove N - ugyanaz mint a **:tabm N**

Ha egy adott parancsot le akarunk futtatni az összes, tabokon nyitott állományban, akkor használjuk a **:tabdo** vagy a **:tabd** parancsot (ugyanazt jelentik). Egy példa rá:
:tabdo %s/foo/bar/g - ez a parancs lecseréli az összes dokumentumban a „foo” minden előfordulását „bar”-ra.

A shell és az MC vim-üzemmódja

Linux alatt a shellünk aktuális beállításait a
set -o
paranccsal listázhatjuk ki. Erre kapunk egy hosszú listát, aminek a vége felé valószínűleg az fog szerepelni, hogy
vi off
Van tehát egy „vi" nevű opciója, mely azonban jelenleg ki van kapcsolva. De ezt be is kapcsolhatjuk, a
set -o vi
paranccsal.
Sőt, ha ezt beírjuk a **.bashrc** (vagy, ha a shellünk a zsh, akkor a **.zshrc**) fájlba, akkor már minden induláskor így fog működni. Ami azt jelenti, hogy a parancssor is „vi(m) üzemmódba" kerül. Tehát az épp szerkesztett (gépelt) sort a Vim szokásos billentyűkombinációival szerkeszthetjük. Az eddig begépelt parancsokat egy láthatatlan fájlként foghatjuk fel, melynek mindig épp csak az egyik sorát láthatjuk. Ez nagyon hasznos, mert így minden parancs begépelésénél gyakorolhatjuk a Vim alapműveleteit. És, ellentétben egyes véleményekkel, továbbra is fog működni a Tab-bal történő automatikus fájlnévkiegészítés!

Ha vissza akarunk térni a „hagyományos" szerkesztési módszerhez, ahhoz a
set -o emacs
parancsot kell kiadnunk.

Ami meg a leggyakoribb, legelterjedtebb fájlkezelőt, az **Mc**-t illeti: ebben van egy belső file-editor, mely az **F4** billentyű lenyomására aktivizálható. Ez azonban nem sok mindent tud. De rávehetjük, hogy ehelyett a Vimet használja: ehhez az kell, hogy a
Beállítások/Alapbeállítások menüben vegyük ki a pipát a
Belső szövegszerkesztő (Mcedit)
sor elől.
Ekkor azt a szövegszerkesztőt fogja használni, amit a $EDITOR változóban definiáltunk neki, s ami valószínűleg már alapesetben is a Vim.

A map parancsok

A Vim kitűnő lehetőséget biztosít arra, hogy „átdefiniáljuk a billentyűzetet”, azaz, hogy „gyorsbillentyűket” hozzunk létre, tehát hogy egyes billentyűket speciális, nekünk tetsző jelentéssel ruházzunk fel. Erre szolgál a map parancs, s ennek különböző változatai.

A map parancsnak ugyanis több fajtája van, azért, mert a Vimnek is több üzemmódja van, s egyes billentyűk jelentése változhat attól függően, hogy éppen milyen üzemmódban nyomtuk le őt. A map parancs változatainak jelentése:

cmap - parancssor
imap - inzert mód
map - normál (parancs), vizuális és operátor mód
map! - parancssoros mód és inzert mód
nmap - normál mód
omap - operátor mód
xmap - vizuális mód
vmap - vizuális és select mód
smap - select mód

Az „operátor mód” azt jelenti, hogy normál módban használtuk a Vim egyik úgynevezett „operátorát”. Operátoroknak számítanak a következő parancsok:
c, d, y, ~, g~, gu, gU, !, =, gq, g?, <, >, zf, g@
(ezek nem mindegyikéről történik említés e könyvben, mely csak bevezetés óhajt lenni).

Ha argumentum nélkül adjuk ki a cmap, imap, stb parancsokat (például így: **:map**) akkor a Vim kiírja, hogy milyen billentyűhozzárendelések érvényesek abban az üzemmódban.

Efféle listát kapunk alul a parancssor fölött:

```
:nmap
n  <C-W>*       * <C-W><C-S>*
n  <C-W>#       * <C-W><C-S>#
n  <F2>         * :lchdir %:p:h<CR>:pwd<CR>
```

Az első oszlopban található karakter azt jelenti, milyen módban működik az adott billentyűkombináció. A lehetséges variációk:

n Normál mód.
i Inzert mód.
v Vizuális és „select” mód.
x Vizuális mód.
s Select mód.
c Parancssoros mód. („**C**ommand line mode”)
o Operátor mód.
<Space> Normál, Vizuális és operátor mód.
! Inzert és parancssoros mód.

Ha a :map (és az :imap meg a többi hasonló) parancs után megadunk egy karaktert is,

akkor az azzal kezdődő parancsokat listázza csak ki. Például a
:nmap g
parancs kilistázza a normál módú, g-vel kezdődő parancsokat.

Egy egyszerű példa a map használatára:
:map <F10> <Esc>:tabnew<CR>
Ez azt eredményezi, hogy az F10 billentyű megnyomása egyenértékű lesz azzal, mintha átmennénk parancs módba (ehhez kell az <Esc>), majd kiadnánk egy :tabnew parancsot, s utána Entert nyomnánk (az Entert jelzi a <CR>). Természetesen a gyakran szükséges billentyűkombinációkat betehetjük a .vimrc fájlunkba, s így azok már rögvest indulás után használhatóak.
Másik példa:
:imap kn királynő - Ennek hatására ha inzert módban leütjük a „kn" karaktersorozatot, azt behelyettesíti a „királynő" szóval. (akkor is, ha egy szó belsejében ütjük le a „kn" karaktersorozatot).

A speciális billentyűk megadására példák:
<C-n> = Ctrl-n
<M-n> = Alt-n
<A-n> = Alt-n
<S-F2> = Shift-F2
<C-Esc> = Ctrl-Esc
<S-A-F2> = Shift-Alt-F2
<BS> = Backspace
<Tab> = Tab
<CR> = Enter
<Enter> = Enter
<Return> = Enter
<Esc> = Escape
<Space> = Space
<Up> = Kurzorfel nyíl
<Down> = Kurzorle nyíl
<Left> = Kurzorbalra nyíl
<Right> = Kurzorjobbra nyíl
<F1> - <F12> = Funkcióbillentyűk 1-től 12-ig
#1, #2..#9,#0 = Funkcióbillentyűk F1-től F9-ig, és a #0 az F10-es
<Insert> = Insert
<Del> = Delete
<Home> = Home
<End> = End
<PageUp> = PageUp
<PageDown> = PageDown

Egy beállított map törlése:
:unmap <F10> - ez a példa törli az F10 billentyűre beállított map hozzárendelést.
Mindenfajta map parancsnak megvan a maga unmap párja, a következő módon:
cmap - cunmap
imap - iunmap

map - unmap
map! - unmap!
nmap - nunmap
omap - ounmap
vmap - vunmap

A map parancsokat arra is használhatjuk, hogy letiltsunk vele bizonyos billentyűket. Erre egy példa:

:nmap <F4> <Nop> - Ez az utasítás letiltja az F4 billentyűt normál módban, azaz ezután a lenyomására semmi sem fog történni.

Egy apró trükk:
:nmap <S-F12> ggVG"+y - E parancs után a Shift-F12 billentyűkombináció kijelöli és a vágólapra másolja a teljes dokumentumot, hogy azután külső programba be tudjuk illeszteni. A kurzor a dokumentum első sorára mozdul el.

Vannak úgynevezett „noremap" parancsok is. Minden *map*-parancsnak megvan a maga *noremap* párja:

cmap - cnoremap
imap - inoremap
map - noremap
map! - noremap!
nmap - nnoremap
omap - onoremap
xmap - xnoremap
vmap - vnoremap
smap - snoremap

A *noremap*-parancsok ugyanazok, mint a közönséges map-parancsok, de nem rekurzív módon. Ez azt jelenti, hogy az az átírt parancsokat nem mapként értelmezi, hanem konkrét parancsként. Például közönséges map parancs esetében ha van egy ilyen mapunk:

map b c
map c :echo 'b'<CR>

akkor a **b** map hatására a **c** mapet futtatjuk le, de mivel nem noremap, ezért a vim lefuttatja a **c** map-et ami kiirja hogy 'b'.
Ebben az esetben azonban:

noremap b c

a **b** map hatására a **c** parancsot futtatjuk mivel noremap ezért a **c**-t mint vim parancsot értelmezi és vár az intervallumra (pl, „**w**", vagy „**$**"). Tehát ebben az esetben a „map c" nem fut le.

Rövidítések (abbreviációk)

A rövidítések hasonló szerepet töltenek be, mint a :map parancs, de általában nem egyetlen karakterre vonatkoznak, hanem egy karaktersorozatra. Az egészet rögvest megvilágítja egy példa:

:ab ft fényestekintetű

Ez a példa-parancs azt eredményezi, hogy ha beütjük az ft karaktersorozatot, majd utána szóközt vagy Entert nyomunk, vagy más olyan karaktert ami nem betű vagy szám, akkor azt kicseréli a „fényestekintetű" szóra. Ez – ellentétben a **:map** paranccsal – nem működik szavak belsejében, tehát ha az „aft" karaktersorozatot ütjük be, annak „ft" végét nem cseréli le. Ez a helyettesítés működik a keresés (*/*) és csere (**:s**) parancs argumentumai esetében is. Ha olyan rövidítést akarunk csinálni, mely csak inzert módban működik, akkor az **:iab** parancsot használjuk.

:ab - kilistázza a rövidítéseket

:una X - az X rövidítés törlése. Az „X" tetszőleges rövidítés lehet.

Létezik még a **:ca** és a **:cuna** parancs is, ami ugyanaz mint az :ab illetve az :una, de csak a parancssorban használható. Létezik továbbá **:ia** és **:iuna** parancs is, az előzőeknek a párja, de inzert módban.

:abc - minden rövidítést töröl.

:iabc - töröl minden, csak inzert módban használható rövidítést.

:cabc - töröl minden, csak parancssorban használható rövidítést.

A fold parancsok

A Vim lehetőséget biztosít rá, hogy a dokumentum egyes részeit mintegy „bezárjuk", „betömörítsünk" egyetlen sorrá. Egy-egy ilyen „bezárt" szöveg neve angolul a „fold". Ahhoz hogy ezt megértsük, legjobb kipróbálni: jelöljünk ki vizuális módban (Ctrl-v) több egymás utáni sort, majd adjuk ki a

zf

parancsot. Ekkor a kijelölt sorok eltűnnek, s a helyükön egyetlen sor látszik majd, amibe (angolul) ki van írva hogy e fold hány „eredeti" sort tartalmaz épp, valamint ezután szerepel a foldban levő sorok közül az elsőnek a szövege.
Ha a kurzorral egy foldot jelző soron állunk épp, azt „kinyithatjuk" a

zo

paranccsal. A Vim azonban emlékezni fog rá, hogy e sorok egy foldhoz tartoznak, s így ha a foldhoz tartozó sorok bármelyikén is állunk a kurzorral, úgy a kinyitott foldot bezárhatjuk újra a

zc

paranccsal.

A foldok használata nagyon hasznos, ha csak gyorsan át akarunk tekinteni egy hosszú dokumentumot, de annak egyes részeire nem vagyunk kíváncsiak.

Hasznos fold-parancsok:

zR - kinyit minden foldot
zf1G - mindent sorba zár a dokumentum elejétől a kurzor soráig
zfG - mindent foldba zár a kurzor sorától a dokumentum végéig
[z - az aktuális (a kurzort tartalmazó) nyitott fold elejére ugrik
]z - az aktuális (a kurzort tartalmazó) nyitott fold végére ugrik
zj - a következő fold elejére ugrik
zk - az előző fold végére ugrik
zc - ha a kurzor egy foldban áll, úgy törli azt a foldot. Ez nem a sorok törlését jelenti, hanem hogy a Vim „elfelejti", hogy azok a sorok egy foldhoz tartoznak
zf'a - az „a" könyvjelzőig mindent foldba zár

A szabályos kifejezések[3]

A „szabályos kifejezés”, angolul „regular expression”, röviden „regexp” (magyarra a „reguláris kifejezés” szókapcsolattal is szokták lefordítani) a Vim egy nagyon fontos alkotóeleme. Mint azt már az előző fejezetekben is megemlítettük, használhatóak a keresés („*/* ”) és csere („**:s**”) parancsok esetén, és hihetetlen mértékben megnövelik munkánk hatékonyságát. Lássuk, milyen elemekből áll össze egy szabályos kifejezés!

A szabályos kifejezés betűket és különleges karaktereket tartalmaz. A különleges karaktereknek legtöbbször speciális jelentése van a szabályos kifejezésben. Egy példa erre: **^[abc]** - e példa azokat a kifejezéseket jelenti, melyek első betűje „a”, „b”, vagy „c” karakterrel kezdődik.

Ha két vagy több szabályos kifejezést egymás után írunk, akkor az eredmény maga is szabályos kifejezés lesz.

Ha a szabályos kifejezésben egy közönséges karakter szerepel, akkor az „illeszkedik önmagára”, tehát „önmagát jelenti”. Közönséges karakternek számít minden, ami nem ***** , **.**, **[**, **** , **]**, **^** , vagy **$**, ezek ugyanis a szabályos kifejezésekben különleges jelentéssel bírnak. Lássuk ezek jelentését!

—A **** karakter:
Ha a **** utáni karakternek valami különleges jelentése van, akkor a „\” kikapcsolja azt, s az utána következő karaktert a Vim közönséges karakterként értelmezi. Ha tehát valamiben például a „***** ” karakterre akarunk rákeresni, akkor a ***** kifejezést kell használnunk.

—A **^** karakter:
A mintát a sor elejére igazítja. Ez azt jelenti, hogy csak azok a kifejezések illeszkednek rá, amelyek a „^” jel utáni szabályos kifejezésnek megfelelnek, de úgy, hogy az illeszkedő rész éppen *a sor elején* kezdődik. Szóköz sem lehet előtte!

—A **$** karakter:
Jelentése hasonló a „^” karakteréhez, de a mintát *a sor végére* igazítja.

—A **.** (pont) karakter:
Az újsor kivételével minden karakter illeszkedik rá. Vagyis ez jelenti a „bármiféle” karaktert. Egy pont csak egy karakterre illeszkedik! Ha tehát 3 egymás utáni tetszőleges karaktert akarunk keresni, akkor 3 pontot kell írnunk így: **...**
Ha ellenben ténylegesen három egymás utáni pontot keresünk, azt így kell írnunk: **\.\.\.**

—A **[ ]** karakterek, vagyis a szögletes zárójelek használata:
A szögletes zárójelbe tett karakterek bármelyike illeszkedik rá. A szabályos kifejezésben ez tehát egyetlen karakternek felel meg, de ez bármelyik lehet a szögletes zárójelben megadottak közül. Például az **[abc][def]** kifejezés tehát illeszkedik a következő találatok bármelyikére: ad, ae, af, bd, be, bf, cd, ce, cf.

Ha a zárójelen belül a ^ jelet használjuk akkor az az előző forma tagadása, vagyis a szögletes zárójelen belül levő karakterek KIVÉTELÉVEL bármelyik karakter illeszkedik rá. Példa: **[^abc]** illeszkedik bármelyik karakterre, KIVÉVE az „a”, a „b” és a „c” karaktert.

A szögletes zárójelen belül karaktertartományt is megadhatunk a következő formában:

3 E fejezethez felhasználtam *Büki András: Unix/Linux héjprogramozás* című könyve megfelelő fejezetének anyagát is.

[tól-ig]
A megadott tartományon belül eső karakterek bármelyike illeszkedik rá. A tartományt az angol abc betűsorrendje szerint kell érteni. Például a [0-9] csak a számokra illeszkedik, az [a-z] a kisbetűkre, az [A-Z] a nagybetűkre, az [a-zA-Z] a betűkre, a [0-9a-fA-F] tetszőleges hexadecimális számjegyre.

—A * karakter:
Ez az „ismétlő karakter". A csillag előtt álló karakter akárhány előfordulása illeszkedik rá, *a nulla darab előfordulás is*!
Ha tehát például keressük azokat az előfordulásokat, amelyek „a" betűvel kezdődnek, s ahol az „a" betűt nulla vagy akárhány tetszőleges karakter követ, azt így kell írnunk: **a.***

—A **+** karakter:
Ez is ismétlő karakter, de – ellentétben a *-gal – ez megköveteli, hogy az őt megelőző karakter vagy kifejezés legalább egyszer előforduljon. Ha tehát keressük azokat a találatokat, melyek egy darab tetszőleges betűvel kezdődnek, melyet tetszőleges számú bármilyen számjegy követ, azt így kell írnunk: **[a-zA-Z][0-9]\+**
A „\" jel e példában amiatt van a „**+**" jel előtt, mert „bekapcsolja" a „+" jel speciális jelentését, enélkül ugyanis a „+" jelet csak közönséges karakterként értelmezné.

—A **( )** karakterek, vagyis a kerek zárójelek használata:
A kerek zárójelek segítségével képezhetünk csoportokat a szabályos kifejezésekből. A zárójelbe tett csoportok egy egységként lesznek kezelve, és a jelentésmódosító jelek a teljes csoportra érvényesek. Ha például olyan előfordulásokat keresünk a szövegben, melyekben az „ab" betűpáros legalább egyszer szerepel, a következő kifejezést kell használnunk: **\(ab\)\+**
A „\" jel e példában amiatt van a kerek zárójelek előtt, mert „bekapcsolja" a kerek zárójelek speciális jelentését, enélkül ugyanis a zárójeleket csak közönséges karakterekként értelmezné.

—A **{ }** karakterek, vagyis a kapcsos zárójelek használata:
Ezzel az operátorral az illeszkedések pontos számát adhatjuk meg. Ha a kapcsos zárójelek közt csak egy szám szerepel, akkor *pontosan* ennyiszeres illeszkedést várunk. Ha a szám után egy vessző is van, akkor *legalább* ennyiszeres az illeszkedés. Ha pedig két számot adunk meg vesszővel elválasztva, akkor legalább az első, de legfeljebb a második által meghatározott többszörözés érvényes. Keressük meg például azokat a helyeket a szövegben, ahol a sorok végén pontosan 3 pont van (befejezetlen mondatok): **\(\.\{3\}\)$**
Legalább négy, egymást követő pontot tartalmazó előfordulások (űrlapon kitöltendő mezők): **\.\{4,\}**
Legalább 3, de legfeljebb 7 számot tartalmazó előfordulások: **[0-9]\{3,7\}**
A „\" jel e példákban amiatt van a kapcsos zárójelek előtt, mert „bekapcsolja" a kapcsos zárójelek speciális jelentését, enélkül ugyanis a zárójeleket csak közönséges karakterekként értelmezné.

Megjegyzendő, hogy a **\{0,\}** jelentése azonos a „*" jelentésmódosító karakter jelentésével (nulla vagy több illeszkedés), a **\{1,\}** pedig a „**+**" megfelelője (egy vagy több illeszkedés). A **\{1\}** pedig azt jelenti, hogy „pontosan egy illeszkedés".

Egy szabályos kifejezés a Vimben úgynevezett „atomokból” áll össze. Egy atom egy vagy néhány karakter sorozatából áll, melyeknek speciális jelentésük van. Lássuk, melyek ezek!

^ A minta elején vagy a „\|”, „\(„, „\%(„ és „\n” után: a sorkezdetet jelenti; Egyéb pozícióban pontosan a '**^** ' karaktert jelöli.

\ ^ Pontosan a '**^** ' karaktert jelöli. E jelölés a minta bármely pozícióján használható.

_^ Sorkezdetre illeszkedik. E jelölés a minta bármely pozícióján használható. Példa: a **_s*_^foo** jelentése: akárhány szóköz vagy újsor, és utána a „foo” egy sor elején.

$** A minta végén vagy a „\|”, „\)” illetve „\n” elején: a sorvégre illeszkedik <EOL>; Egyéb pozícióban pontosan a '**$' jelet jelenti.

\$ Pontosan a '**$**' jelet jelenti. E jelölés a minta bármely pozícióján használható.

$** Sorvéget jelöl. E jelölés a minta bármely pozícióján használható. Megjegyzendő, hogy a "**a_$b" sohasem illeszkedik. Ehelyett az "**a\nb**" kifejezést kell használni. Példa: a **foo$_s*** jelentése: „foo” a sor végén és ezt követi akárhány whitespace karakter vagy újsor.

. Illeszkedik bármely közönséges karakterre vagy sorvégre. Legyünk óvatosak, mert a „**.***” illeszkedik a teljes szövegre a buffer végéig!

\< Egy szó kezdetére illeszkedik: a következő karakter egy szó első karaktere.

\> Egy szó végére illeszkedik: az előző karakter egy szó utolsó karaktere.

\zs Bármely pozíción használható, s erre a pozícióra állítja az illeszkedés kezdetét: a következő karakter az illeszkedés első karaktere. Példa: **/^\s*\zsif** jelentése: sorkezdet, utána akárhány whitespace, majd az „if” karaktersorozat. Fontos megértenünk a különbséget az előbbi példa és a /^\s*if változat között. Ez utóbbi ugyanis nemcsak az „if” karaktersorozatot tartja a találatnak, hanem a találat részének tekinti az azt megelőző whitespacékat is!

\ze Bármely pozíción használható, s erre a pozícióra állítja az illeszkedés végét: az előző karakter az illeszkedés utolsó karaktere. Példa: az **end\ze\(if\|for\)** illeszkedik az „end”-re az „endif”-ben és az „endfor”-ban.

\%^ Illeszkedik a fájl elejére.

\%$ Illeszkedik a fájl végére.

\%V A „vizuális módban” kiválasztott területre illeszkedik. Ha a vizuális mód már véget ért (nincs aktuális kiválasztás kijelölve) akkor arra a területre illeszkedik, amit a **gv** parancs újra kiválasztana.

\%# A kurzor pozíciójára illeszkedik.

\%'m Az „m” könyvjelzővel megjelölt pozícióra illeszkedik.

\%<'m Az „m” könyvjelzővel megjelölt pozíció előtti pozícióra illeszkedik.

\%>'m Az „m” könyvjelzővel megjelölt pozíció utáni pozícióra illeszkedik.

\%23l A meghatározott sorszámú sorra illeszkedik.

\ % < 2 3 l A meghatározott sorszámú sor fölötti sorokra illeszkedik.

\ % > 2 3 l A meghatározott sorszámú sor alatti sorokra illeszkedik.

A fenti 3 példában a „23” helyén természetesen bármilyen sorszám állhat.

\ % 2 3 c Illeszkedik a meghatározott oszlopra.

\ % < 2 3 c Illeszkedik a meghatározott oszlop előtti oszlopokra.

\ % > 2 3 c Illeszkedik a meghatározott oszlop utáni oszlopokra.

A fenti 3 példában a „23” helyén természetesen bármilyen oszlopszám állhat.

Karakterosztályok:

\ p Illeszkedik a nyomtatható karakterekre.

\ P Mint „\p”, de kivéve a számjegyeket.

\ s Whitespace karakter: <Space> és <Tab>

\ S Nem-whitespace karakter; az „**\ s** ” ellentéte.

\ d Számjegy: [0-9]

\ D Nem-számjegy: [^0-9]

\ x Hexadecimális számjegy: [0-9A-Fa-f]

\ X Nem hexadecimális számjegy: [^0-9A-Fa-f]

\ o Oktális számjegy: [0-7]

\ O Nem oktális számjegy: [^0-7]

\ w Szókarakter: [0-9A-Za-z_]

\ W Nem szókarakter: [^0-9A-Za-z_]

\ h Szókezdő karakter: [A-Za-z_]

\ H Nem szókezdő karakter: [^A-Za-z_]

\ a Alfabetikus karakter: [A-Za-z]

\ A Nem alfabetikus karakter: [^A-Za-z]

\ l Kisbetű: [a-z]

\ L Nem kisbetű: [^a-z]

\ u Nagybetű: [A-Z]

\ U Nem nagybetű: [^A-Z]

A fentebb megadott karakterosztályok mindegyikének létezik egy párja, amit az aláhúzásjel előz meg, tehát _p, _P, _s, _S, _d, _D, _x, _X, _o, _O, _w, _W, _h, _H, _a, _A, _l, _L, _u, _U

Ezek ugyanazokat a karaktereket jelentik, mint az aláhúzásjel nélküli párjaik, de a sorvégjellel (<EOL>) kiegészítve.

\ e Illeszkedik az <Esc>-re

\t Illeszkedik a <Tab>-ra

\r Illeszkedik a <CR>-re

\b Illeszkedik a <BS>-re

\n Illeszkedik a sorvégre

~ Illeszkedik az utolsó behelyettesített stringre

\1 Ugyanarra a stringre illeszkedik, amely illeszkedett az első alkifejezésre is. Példa: a **\([a-z]\).\1** illeszkedik az ilyesféle szavakra: „ata”, „ehe”, „tot”, stb.

\2 Mint „**\1**”, de a második alkifejezésre.

...

\9 Mint „**\1**”, de a kilencedik alkifejezésre.

x Egy egyszerű karakter, speciális jelentés nélkül, mindig önmagát jelenti.

\x Egy backslasht követő egyszerű karakter, speciális jelentés nélkül, későbbi bővítésre fenntartva.

A global parancs

A *global* parancs általános formája:
g/kifejezés/parancs - végrehajtja a „parancs”-ot a fájl minden olyan során, melyre illeszkedik a „kifejezés”.
g!/kifejezés/parancs - végrehajtja a „parancs”-ot a fájl minden olyan során, melyre NEM illeszkedik a „kifejezés”.
v/kifejezés/parancs - ugyanaz, mint a **g!**
Példa:
:g/pat/s//PAT/g
A fenti példa a „pat” minden előfordulását lecseréli „PAT”-ra. Azaz ugyanaz, mintha ezt a parancsot adnánk ki:
:%s/pat/PAT/g

A *global* parancsnak tartományt is megadhatunk, ekkor csak a tartományon belüli sorokban keres. Példák:
:1,15g/Peti/ - kilistázza azokat a sorokat, melyek az 1-es és 15-ös sorok közé esnek, és szerepel bennük a „Peti” szó.
:15,$g/Peti/ - kilistázza azokat a sorokat, melyek a 15-ös sor és a dokumentum vége közé esnek, és szerepel bennük a „Peti” szó.

2. rész

Tippek és trükkök

Kalkulátorkészítés

Vegyük fel a .vimrc-be a következő két sort:

:command! -nargs=+ Calc :py print <args>
:py from math import *

Ezek után (miután újraindítottuk a Vimet) a parancssora kalkulátorként is használható: például a **:Calc 2*5+6** parancsra kiírja, hogy 16. De függvényeket is számolhatunk vele, például azt, hogy **:Calc sin(pi/5)**

Megjegyzés: a Calc szó muszáj hogy nagybetűvel kezdődjék! Továbbá, ehhez a Vim Python-supporttal kell hogy lefordított legyen.

Ha csak egész számokkal akarunk számolni, másként is megoldhatjuk a kalkulátorkérdést. Üssük le a **"** (idézőjel) gombot, majd az **=** jelet, mire a parancssorba kerülünk, melynek elejére ki is írja a Vim az = karaktert. Ezután be kell írnunk a kiszámítandó kifejezést, majd Entert nyomni. Látszatra semmi sem történik, csak kilépünk a parancssorból, de a **p** gomb lenyomására az eredmény beíródik a kurzor aktuális pozíciójába.

Karakterek Ascii kódjának kijelzése

Vegyük fel a következő sort a .vimrc fájlba:

set statusline=%<%f%h%m%r%=%b\ 0x%B\ \ %l,%c%V\ %P

Ez azt fogja eredményezni, hogy bármiként is mozgunk a szövegben, a Vim a státuszsorban, jobboldalt lent mindig kijelzi (az aktuális kurzorpozíció előtt) a kurzor alatti karakter Ascii kódját decimálisan és hexadecimálisan. A fenti sor részletes megértéséhez tanulmányozd a leírást, amihez a **:h statusline** paranccsal férhetsz hozzá! Ott még sok érdekes opciót találhatsz.

Beillesztés nem lépcsőzetesen eltolva

Sajnos ha valami programrészletet másolunk a terminálokból vagy a böngészőnk egy ablakából a Vimbe, az – úgy tapasztaltam – furán lesz beszúrva – az egyes programsorok kezdetei mindinkább jobbra csúsznak. Nem tudom, mi ennek az oka; mindenesetre ez elkerülhető, ha másolás előtt kiadjuk a
:set paste parancsot. A visszatérés a „normál” viselkedéshez elérhető a
:set nopaste paranccsal.

Vegyük fel a következő sort a .vimrc fájlba:
:map <F12> :set invpaste<CR> - ez azt fogja eredményezni, hogy az F12 gomb megnyomására vált a „paste” és „nopaste” módok között. Megjegyzendő, hogy e gomb nem működik inzert módban, csak parancsmódban.

A státuszsor színének változtatása üzemmódtól függően

Vegyük be a következő kis kódrészletet a .vimrc állományba:

```
" Most beállítjuk a státuszsor színét az aktuális üzemmódtól függően
" Normál (parancs)mód = zöld, Inzert mód = magenta
if version >= 700
  au InsertEnter * hi StatusLine term=reverse ctermbg=5 gui=undercurl guisp=Magenta
  au InsertLeave * hi StatusLine term=reverse ctermfg=0 ctermbg=2 gui=bold,reverse
endif
```

Ennek hatására a státuszsor az éppen használt módtól függően fogja a színét változtatni, automatikusan: Normál módban zöld lesz, inzert módban magenta.

Munka különböző fájlformátumokkal

Különböző operációs rendszerek különbözőképpen jelölik a sorvégeket: az Unix/Linux „LineFeed”-del (LF), a Microsoft Windows „CarriageReturn”-nal amit „LineFeed” követ (CRLF), a Mac pedig „CarriageReturn”-nal (CR). Ez problémákat okozhat. Szerencsére a Vim a fájlok megnyitásánál detektálja – általában sikeresen – az adott formátumot, és alkalmazkodik hozzá. A kívánt formátumot azonban mi is beállíthatjuk, a

:set ff=format

paranccsal, ahol a ***format*** értéke ***dos*** a Microsoft Windows/DOS fájloknál, ***unix*** az Unix/Linux rendszereknél, és ***mac*** az Apple Mac esetében. E parancs után természetesen el kell menteni a fájlunkat, hogy a formátumváltás maradandó legyen.

Ha a beolvasott fájl esetében a Vim nem találta volna el a kellő formátumot, akkor kierőszakolhatjuk nála a kívánságunk szerinti formátumot az

:e ++ff=format

paranccsal.

Áramszünet...

Mi történik, ha egy áramszünet (vagy más ok) miatt „szabálytalanul lett kilépve” a Vimből? Ekkor az történik, hogy a következő indításnál a Vim figyelmeztető üzenetet küld nekünk efféleképp (e példában a megnyitni kívánt file neve „proba.txt” volt):

```
E325: ATTENTION
Found a swap file by the name ".proba.txt.swp"
          owned by: vz   dated: Tue May 31 07:13:15 2011
         file name: ~vz/proba.txt
          modified: no
         user name: vz   host name: Csiszilla
        process ID: 3147
While opening file "proba.txt"
             dated: Tue May 31 11:15:16 2011
      NEWER than swap file!

(1) Another program may be editing the same file.
    If this is the case, be careful not to end up with two
    different instances of the same file when making changes.
    Quit, or continue with caution.

(2) An edit session for this file crashed.
    If this is the case, use ":recover" or "vim -r proba.txt"
    to recover the changes (see ":help recovery").
    If you did this already, delete the swap file ".proba.txt.swp"
    to avoid this message.

Swap file ".proba.txt.swp" already exists!
[O]pen Read-Only, (E)dit anyway, (R)ecover, (D)elete it, (Q)uit, (A)bort:
```

Ez magyarul nagyjából annyit jelent, hogy a Vim talált egy a megnyitni kívánt állományhoz tartozó swap fájlt, tehát vagy
1.Épp egy másik program használja a fájlt, ekkor jobb ha nem piszkáljuk, vagy
2.Valamiért szerkesztés közben „crashed” történt, azaz a szerkesztés szabálytalanul ért véget. Épp ez az eset áramszünetkor. Mindenesetre több választási lehetőséget kínál fel nekünk: megnyithatjuk „csak olvasható” módban (`[O]pen Read-Only`), vagy szerkeszthetjük mindenáron (`(E)dit anyway`), helyreállíthatjuk a fájlt (`(R)ecover`), törölhetjük a swap fájlt (`(D)elete it`), vagy kiléphetünk (`(Q)uit, (A)bort`).

Ha biztosak vagyunk benne, hogy áramszünet történt, más progi nem használja a fájlt, s így a swap nem tartalmaz semmi értékeset, akkor válasszuk a „Delete” opciót a „D” leütésével.

A képernyő nevének megváltoztatása

Megváltoztathatjuk a képernyő nevét (*window title*) valami nekünk tetszőre a következő módon:
:set title titlestring=EzAzÚjNév
Ha a kívánt képernyőnév szóközt is tartalmaz, azt így kell megadni:
:set title titlestring=Ez\ az\ új\ név

Az alapértelmezett képernyőnév az állománynév, melyet egy olyan karakter követ, mely jelzi az állomány állapotát, s ezt követi zárójelek közt az állomány tartalomjegyzékének a neve. Ez általában a **(~)**. Az állomány állapotát jelölő karakter a következők egyike lehet:

- Az állomány nem módosult
+ Az állomány módosult
= Az állomány csak olvasható
=+ Az állomány csak olvasható, és módosult

Számozások

Előfordulhat, hogy be akarjuk számozni a sorokat. Ha tegyük fel 1-től 10-ig akarunk számozott sorokat készíteni, azt megtehetjük ezzel a paranccsal:

:for i in range(1,10) | put =i | endfor

Vagy itt van egy kissé bonyolultabb példa:

:for i in range(1,10) | put ='192.168.0.'.i | endfor

E fenti példa a következő outputot eredményezi:

```
192.168.0.1
192.168.0.2
192.168.0.3
192.168.0.4
192.168.0.5
192.168.0.6
192.168.0.7
192.168.0.8
192.168.0.9
192.168.0.10
```

Ez pedig:

:for i in range(1,10) | put =i.'.192.168.0.'.i | endfor

a következőt:

```
1.192.168.0.1
2.192.168.0.2
3.192.168.0.3
4.192.168.0.4
5.192.168.0.5
6.192.168.0.6
7.192.168.0.7
8.192.168.0.8
9.192.168.0.9
10.192.168.0.10
```

Saját parancsok definiálása

Készíthetünk saját parancsokat, melyeket a parancssorban (*command line*) használunk. Erre szolgál a **:command** parancs (a rövidebb **:com** forma is alkalmazható). Használata:

:command - azaz paraméter nélkül használva kilistázza az összes, általunk definiált parancsot.
:command *string* - kilistázza azokat az általunk definiált parancsokat, melyek a megadott stringgel kezdődnek.
:command *név parancs* - saját parancs definiálása „név" néven. A „parancs" bármi lehet, amit szokásosan beírhatunk a parancssorba. Megjegyzendő, hogy a „név"-nek nagy kezdőbetűsnek kell lennie! Valamint fontos tudni azt is, hogy ha a megadott nevű parancs már létezik, akkor hibaüzenetet kapunk. Ezért, ha egy parancsot át akarunk definiálni, használjuk a **:command!** formát.

Példa:
:command For for i in range(1,10) | put =i | endfor
Ezután a **:For** parancs létrehoz nekünk tíz, 1-től 10-ig számozott sort.
Másik példa:
:command Ls !ls -all %
A fenti definíció azt eredményezi, hogy az **:Ls** parancs megjeleníti nekünk az aktuális dokumentum nevét, attributumait, egyéb tulajdonságait, ahogy azt a Linux alatt az ls parancs szokta.

Olyan parancsot is készíthetünk, mely paramétereket (argumentumokat) is elfogad. Ennek formátuma:
:command -nargs=spec név parancs ahol a „spec" a következők egyike lehet:

spec	*jelentése*
1	a parancs pontosan egy argumentumot vár
*	a parancs akárhány argumentumot elfogad
?	semmi vagy egy argumentum
+	egy vagy több argumentum

Egy hasznos példa: Vegyük fel a következő sorokat a **.vimrc** fájlunkba[4]:

```
" Definiáljuk a Sok nevű függvényt ami két paramétert vár:
" 1. darab nevű: hány darab karakter kell kiírni
" 2. karakter nevű: ezt a karaktert kell kiírni
function! Sok(darab, karakter)
 let s:sok_karakter = '' " Értékadás, itt inicializálás.
" Az s: prefix használatával lesz script szinten lokális a változó, egyébként globális lenne.
   for i in range(1, a:darab) " a függvénytörzsben az a:paraméternévvel érjük el
" az aktuális paramétert.
" A . operátor a hozzáfűzés, a .= a rövidebb forma a
" let s:sok_karakter = s:sok_karakter . a:karakter helyett
  let s:sok_karakter .= a:karakter
   endfor
put =s:sok_karakter " Az egész karaktersorozatot kiírjuk
```

4 A következő „Sok" nevű függvényt Kalmár Zoltán (kalmiz@gmail.com) készítette, akinek ezúton szeretnék köszönetet mondani.

```
  unlet s:sok_karakter " Változó törlése
  endfunction
" Ha meghívjuk a ":Sok 42 a" parancsot akkor az <f-args> következtében
" olyan mintha ezt írnánk:
" call Sok('42', 'a')
command! -nargs=* Sok call Sok(<f-args>)
```

Ezután a Vimünk ismerni fogja a „Sok” parancsot, melynek használata a következőképp lehetséges:

:Sok 42 =

Ez kiír egymás után 42 darab egyenlőségjelet a kurzor aktuális sora alá beszúrt sorba, a kiírást a sor elejétől kezdve. A kurzort e sor elejére mozgatja. Ez nagyon hasznos, ha a szövegben sok azonos karakterből álló elválasztóvonalakat akarunk szerepeltetni. E példa a függvények használatát is bemutatja kissé.

Szavak automatikus kiegészítése

Üssünk be inzert módban egypár betűt egy szó elejéből, majd nyomjuk meg a **Ctrl-n** billentyűkombinációt. Ekkor, ha már volt az állományunkban olyan szó amely a beütött karakterekkel kezdődik, a Vim kiegészíti a beütött stringrészletet az adott szóra. Például tegyük fel, hogy volt már olyan szó a szövegben, hogy „királynő”, ekkor ha beütjük a „**kir**” karaktereket majd **Ctrl-n** -et nyomunk, a Vim kiegészíti a „kir”-t „királynő”-re.

Abban az esetben ha több, ugyanúgy kezdődő szó is van a szövegben, a **Ctrl-n** megnyomása után a beütött karakterek alatt inverzben felsorolja a lehetséges megfeleléseket, e kis listában a kurzorbillentyűkkel mozoghatunk, majd amelyiken épp állunk ott Entert nyomva az kerül kiválasztásra.

Az utolsó szerkesztett/beillesztett szöveg vizuális kiválasztása

Vegyük fel a következő sort a .vimrc fájlba:
nmap gV `[v`]
Ennek hatására, ha parancsmódban kiadjuk a **gV** parancsot, a Vim inverzben kijelöli az utolsó szerkesztett/beillesztett szöveget, a kurzort ennek végére pozicionálja, és átvált vizuális módba.
Magyarázat:
A **`[** hatására elugrik a legutolsó szerkesztés első karakterére, a **v** hatására átvált vizuális módba, a **`]** hatására elugrik a legutolsó szerkesztés utolsó karakterére.

Hosszú sorok tördelése

Ha olyan hosszú egy szöveg, hogy nem fér ki a képernyő egy sorába, akkor a Vim több sorba tördeli azt. Alapértelmezés szerint nem problémázik azon, hogy a sor vége szóhatárra esik-e, és így előfordulhat, hogy kettétör egy szót. Ezt azonban elkerülhetjük ha megváltoztatjuk a Vim 3 beállítását. Ehhez nem kell 3 különböző parancs, elég ha felvesszük ezt a sort a .vimrc fájlba:

command! -nargs=* Wrap set wrap linebreak nolist

Ezután, ha kiadjuk a **:Wrap** parancsot, már nem fogja széttördelni a sorvégi szavakat. Visszatérhetünk azonban a régi „szótörős" viselkedéshez a

:set nolinebreak

paranccsal.

Magyarázat az egyes paraméterekhez:

—Ha a „**wrap**" be van állítva (**:set wrap**) akkor a képernyő egy sorába be nem férő szövegeket több sorba töri. Ha a **wrap** nincs beállítva (**:set nowrap**) akkor a hosszú szövegnek csak egy része látható a képernyőn.

—Ha a „**list**" be van állítva (**:set list**) akkor „$" jelekkel jelöli meg a sorok végét. Ha nincs beállítva (**:set nolist**) akkor nem tesz így.

—Ha a „**linebreak**" be van állítva, akkor nem töri meg szó belsejében a sort. Ez az opció nem működik, ha a „**wrap**" ki van kapcsolva, vagy a „**list**" be van kapcsolva.

A file újratöltése más karakterkódolással

Ha a Vim nem ismerte fel jól a file karakterkódolását, ezt közvetlenül is megadhatjuk neki, azaz újratölthetjük más karakterkódolással. Ennek módja:
:e ++enc=<encoding> ahol az <encoding> értéke a megfelelő kódlap neve. Ez lehet például **utf-8**, vagy **cp850**, **cp437**, **latin2**, stb.
Példa:
:e ++enc=latin2

Fájlok összehasonlítása

Ha kíváncsiak vagyunk két fájl közti különbségekre, mondjuk a proba.txt és a proba2.txt nevűek esetében, akkor használjuk a **diffsplit** parancsot. Ez úgy működik, hogy nyissuk meg az egyik fájlt (mondjuk a proba.txt nevűt), majd adjuk ki a következő parancsot:

:diffsplit proba2.txt

Ez vízszintesen kettéosztja majd a képernyőt, s a felső részében megjeleníti a proba2.txt fájlt. Mindkét fájl esetében szépen kiszínezi a különbségeket: Ha egy sor megvan mindkét fájlban, de mindegyikben egy „kicsit" másként, azaz van pár karakter különbség, akkor az a sor rózsaszín lesz, a plusz karakterek meg vörösek. Ha egy sor megvan a fájlban, de nincs meg a másikban, akkor az kék lesz, s a másik fájlban erre a helyre a Vim csupa mínuszjelekből álló sort ír ki, türkizkék színnel.

Ha a fájlok összehasonlítását nem vízszintesen, hanem függőlegesen felosztott képernyővel szeretnénk elvégezni, akkor a

:vert diffsplit proba2.txt

parancsot használjuk. Ezesetben a (példabeli) proba2.txt file a bal oldalon kerül megnyitásra.

Jó tudni, hogy a diffsplit parancs használata után, ha scrollozzuk az egyik képernyőt, automatikusan scrollozódik a másik is, kivéve, ha ezt letiltjuk a

:set noscrollbind paranccsal.

A fájlok közt megjelölt különbségek között könnyedén mozoghatunk két parancs segítségével:

[c - az előző különbséghez ugrik
]c - a következő különbséghez ugrik

A képernyő színeinek megváltoztatása

Nem tetszenek neked a Vim színei? Sebaj, megváltoztathatod őket! A következő parancs:
:highlight Normal ctermfg=black ctermbg=yellow
sárga háttérszínre ír feketével. Ez pedig:
:highlight Normal ctermfg=green ctermbg=black
fekete alapra ír zölddel – tiszta „Mátrixos" érzés!
A parancsban természetesen a „ctermfg" jelenti a betűszínt és a „ctermbg" a háttérszínt. A „Normal" azt jelenti, hogy a színmeghatározások ezesetben a normál szövegre vonatkoznak. Egyéb lehetőségek:

Cursor - a kurzor alatti karakter színei
ErrorMsg - a parancssori hibaüzenetek színei
Visual - a vizuális módban kiválasztott szöveg színe

A parancsot természetesen bevehetjük a .vimrc fájlba is, hogy már megnyitáskor is így jelenjék meg a dokumentum.