A Vim használata C programozóknak

Ahhoz, hogy kényelmesen tudjunk fejleszteni, nincs feltétlenül szükségünk egy integrált fejlesztőkörnyezetre. Az automatikus kiegészítéstől a ctags és a make meghívásáig a Vim-nek számos olyan szolgáltatása van, ami jelentősen megkönnyítheti egy C-ben fejlesztő programozó életét.

A *Vim* egy olyan rendkívül hatékony szövegszerkesztő, amely ugyan *Bill Joy* immár 30 éves vi szerkesztőjén alapul, de számos új és kellemes szolgáltatása is van. Ugyanakkor azok a szolgáltatások amelyek oly vonzóvá teszik a *Vim*-et mindazok számára, akik már ismerik, gyakorta elriasztják a kezdőket. Ennek a cikknek alapvetően az a célja, hogy segítsen a vállalkozó szelleműeknek a program megismerésében, különös tekintettel a *C* programozással kapcsolatos szolgáltatásokra.

A make parancs, és a fordít-tesztelszerkeszt ciklus

A programozók rendszerint úgy dolgoznak, hogy lefordítják a félkész anyagot, kipróbálják, majd addig szerkesztgetik újra és újra, amíg a kipróbálás során képes nem lesz azt a műveletsort végrehajtani, amire tervezték. Nyilván minden olyan eljárás, ami csökkenti az ezzel a ciklussal kapcsolatos műveletek számát, megkönnyíti a programozó életét. A *Vim* éppen ezt teszi, hiszen magából a szerkesztőből hívhatjuk meg a `make` parancsot, illetve futtathatjuk a kész kódot. Ha az aktuális könyvtárba van *Makefile*, elegendő a szerkesztőben kiadni a `:make` parancsot, és máris fordul a program.

Persze könyvtárat is válthatunk, ha épp erre van szükség, mégpedig a `:cd` parancs segítségével. Ha pedig netán elfelejtettük volna, hol is vagyunk éppen, adjuk ki a `:pwd` parancsot, és máris kiderül. Aki *FreeBSD*-t használ, és a `make` helyett a `gmake` programot szeretné használni, annak sincs túl sok extra teendője. Elegendő kiadnia a `:set makeprg=gmake` parancsot. Tegyük fel mármost, hogy paramétereket is szeretnénk átadni a `make`-nek. Mondjuk a fordításhoz e *gcc 2.96*-os változatát szeretnénk használni, nem az alapértelmezettet. Ezt így érhetjük el:

```
:set makeprg=gmake\ \CC=gcc296
```

Most jön a munka dandárja. Meg kell keresnünk az esetleges hibákat, meg kell találnunk a forrásban a nekik megfelelő kódsort, és kijavítani őket. Ha sorszámoztatni szeretnénk a kódot, adjuk ki a `:se nu` parancsot. A sorszámozást kikapcsolni a `:se nonu` paranccsal lehet.

Amikor elkezdjük fordítani a programot, de az nem sikerül, a *Vim* automatikusan az első problémás sorra ugrik. Ha a következő hibához akarunk ugrani, használjuk a `:cn` parancsot. Hasonlóan a `:cfirst` visszaugrik az első hibás sorhoz, míg a `:clast` az utolsóhoz. Ha kijavítottuk az összes hibát, ismét megpróbálkozhatunk a fordítással. A hibalista ismételt megtekintéséhez a `:clist` parancsot kell használnunk. Aki írt már programot, tudhatja, hogy mindez mennyire kényelmes. Ha a hiba javítása közben bele akarunk olvasni egy másik forrásfájlba, mondjuk a *foo.c*-be, adjuk ki az `:e foo.c` parancsot.

A *Vim*-ben számos olyan gyorsítási lehetőség van, amelyekkel elkerülhetjük a sok gépelést. Az egyik közülük az `:e #` parancs, ami az előző fájlra ugrik vissza. Ha látni szeretnénk az össze pillanatnyilag megnyitott fájl listáját, adjuk ki az `:ls` vagy a `:buffers` parancsot.

Ha úgy gondoljuk, hogy már zavaróan sok fájlt tartunk nyitva, és egyeseket be szeretnénk zárni, először szintén az `:ls` parancsot kell használnunk, ami valami ilyesmit jelenít meg:

```
2 #    "newcachain.c"
➥line 5
3 %a   "cachain.c"
➥line 1
```

Ha mondjuk a *newcachain.c* fájl be akarjuk zárni, akkor ezt a `:bd 2` vagy a `:bd newcachain.c` paranccsal tehetjük meg.

Ha olvasunk egy hosszabb kódot, gyakorta előfordul, hogy egy vagy több függvényt szeretnénk gyorsan átugrani, mert azokban nincs semmi pillanatnyilag érdekes. Ezt parancsmódban a `]]` billentyűkombinációval tehetjük meg. Ha visszafelé haladva akarjuk ugyanezt megtenni, használjuk a `[[` parancsot.

Használhatunk könyvjelzőket is bizonyos helyek bejelölésére. Címkeként használhatunk bármilyen kisbetűt. Ha mondjuk meg akarjuk jelölni a program 256-ik sorát, és ez a pontot „b”-nek szeretnénk hívni, álljunk a kurzorral a megfelelő sorra, majd adjuk ki parancsmódban a `mb` parancsot. Ügyeljünk rá, hogy a *Vim* soha nem igazolja vissza a parancsok végrehajtását, csak végrehajtja őket. Ha a könyvjelzők között ugrálva az előzőre szeretnénk visszaállni,

használjuk a `''` (két egyszeres idézőjel) parancsot. A `'a` parancs az a-val jelzett pontra ugrik, és így tovább.
Néha, de különösen ***Makefile***-ok szerkesztése közben jó ha látjuk, hogy egy üres hely szóközökből vagy tabulátorokból áll. Ilyenkor használjuk a `:se list` parancsot, mire a tabulátorok kék `^I` jelekként fognak megjelenni. Van amúgy egy másik módszer is: a `/\t` sárgával jeleníti meg a tabulátorokat.
A programozók gyakran végeznek az egész forrásban kereséseket, illetve globális szöveghelyettesítést. A ***Vim*** ehhez is kiváló támogatást nyújt.
A kereséshez egyszerűen adjuk ki a `/` parancsot, és a program máris a keresett szóra ugrik. Aki jobban szereti az ***emacs*** által használt inkrementális keresést, az adja ki a `:se incsearch` parancsot a keresés megkezdése előtt. Ezt a szolgáltatást később a `:se nois` paranccsal lehet kikapcsolni.
A keresés és csere szintén egyike a ***Vim*** legnagyszerűbb szolgáltatásainak. Ezt a műveletet végrehajthatjuk csupán a kód egy részén, amit a `v` paranccsal jelöltünk ki, megadhatunk neki kezdő és záró sorszámot, sőt kijelölhetünk a szövegben egy tetszőleges téglalap alakú területet is a ***Ctrl-V*** paranccsal.
Ha például az a feladat, hogy a 24. és 56. sorok között az összes `foo`-t cseréljük ki `bar`-ra, a következőt kell tennünk. Először is jelöljük ki a kérdéses területet a `:24,56` paranccsal. (A kijelölt területhez a két határoló sor is hozzátartozik!) Magát a cserét a `s/foo/bar` paranccsal végezhetjük el.
Ügyeljünk azonban rá, hogy a parancs ebben a formájában minden sorban csak a keresett szöveg egy előfordulását cseréli ki. A globális, minden előfordulást érintő cseréhez a `s/foo/bar/g` parancsot kell használnunk. Ha a cserét csak a keresett szövegrész bizonyos előfordulásainál akarjuk elvégezni, alkalmazhatjuk a `s/foo/bar/gc` parancsot is, amely minden egyes csere előtt rákérdez, hogy végrehajthatja-e a műveletet.
Előfordulhat, hogy a keresett karakterlánc része más kulcsszavaknak is. Tegyük fel például, hogy egy `in` nevű változó nevét akarjuk lecserélni valami másra. Ilyenkor nyilván nem szeretnénk, ha a csere a minta nevű változó „megfelelő” részét is érintené. A megoldás ilyenkor az, ha a keresést egész szóra korlátozzuk a következő módon : `/\<in\>/`.
A leggyakoribb az, amikor egy dolog valamennyi előfordulását ki szeretnénk cserélni egy adott fájlon belül. Ezt az `:1,$s/foo/bar/g` vagy a `:%s/foo/bar/g` parancsok valamelyikével tehetjük meg. Ha ugyanezt a műveletet valamennyi megnyitott fájlon akarjuk elvégezni, a `:bufdo %s/foo/bar/g` formát kell használnunk.
A keresés egy másik módja, ha a kurzorral ráállunk a kulcsszó egy előfordulására, majd parancsmódban beütjük a `*` parancsot. Erre a ***Vim*** a kérdéses szó minden előfordulását kiemeli. Ha egy keresést az adott kurzorpozíciótól visszafelé haladva akarunk elvégezni, használjuk a `?` parancsot a `/` helyett.
Ha egy keresés lefutott, a ***Vim*** megjegyzi annak eredményét. Ha tehát újra ugyanarra a szóra akarunk rákeresni, elég beüti a `/` vagy a `?` parancsot, a keresendő szövegre már nincs szükség.
A keresés mellékhatásaként a találatok kiemelve maradnak a képernyőn, ami szerkesztés közben elég idegesítő tud lenni, különösen, ha nincs is már rá szükség. A kiemelések eltüntetéséhez adjuk ki a `:se nohlsearch` vagy a `:nohl` parancsot.
Ha a kettőspont után egy ***Vim*** parancsot gépelünk, bármikor használhatjuk a ***Tab*** billentyűt, mire a program kiegészíti azt. Ha túl kevés az információ a kiegészítéshez, a ***Tab***-ot többször is megnyomhatjuk, mire újabb és újabb javaslatokat kapunk egészen addig, amíg meg nem találjuk az igazit.

A Vim és az Exuberant ctags

Az ***Exuberant ctags*** (lásd az elektronikus forrásokat) egy olyan külső program, amely a ***Vim*** számára a forráskódban való tájékozódást segítő tageket (jelzőket) képes előállítani.
Ha programunk teljes forráskódja egyetlen könyvtárban található, egyszerűen lépjünk be ide, és adjuk ki a következő parancsot:

```
$ ctags .
```

Ez létrehoz egy ***tags*** nevű fájlt, amelyben nevének megfelelően ezek a bizonyos tag-ek találhatók. Ezt az információt a ***Vim*** is értelmezi, és segítségével villámgyorsan képes odaugrani a különböző függvények, felsorolt típusok meghatározásához, az előfeldolgozónak átadott konstansok és makrók definíciójához (`#define`) és számos más a ***C*** nyelvre jellemző szerkezethez.
Ha a forráskód több könyvtárban szétszórva található, a ***ctags*** ezek helyét is bele kell foglalja az általa előállított információcsomagba. Ilyenkor lépjünk be a forráskód legfelsőbb szintű könyvtárába, és adjuk ki a

```
$ ctags -R .
```

parancsot, majd ellenőrizzük, hogy valóban létrejött-e a ***tags*** fájl. Ez utóbbit egyébként magával a ***Vim***-mel is megnyithatjuk.
Nos, most hogy már tag-jeink is vannak, lássuk, miként használhatjuk őket a forráskódban való mozgás során. A ***ctags*** segítségével való tájékozódás egyike a legnagyszerűbb dolgoknak, amik egy programozó rendelkezésére állnak.
Aki megtanulja használni ezt a szolgáltatást, olyan könnyen és gyorsan tud navigálni, hogy utóbb el se tudja képzelni, hogy tudott eddig meglenni nélküle.
Ha tehát létrehoztuk a ***tags*** fájlt, nyissunk meg próbaképpen egy tetszőleges forráskódot. Az egyetlen eltérés az eddigiektől az, hogy ha a teljes forrás több könyvtárban található, a kérdéses fájl pedig nem a gyökérkönyvtárban van, akkor is innen indulva kell megnyitnunk. Tegyük fel például, hogy forráskódunk könyvtárszerkezete a következőképpen néz ki:

```
common
  |
  ----> gui --> wxpython
  |       |
  |       ------>Tk
  |
  ----> backend -->
networking
  include
  user
```

Ha a ***common/backend/networking*** könyvtárban található ***tcp.c*** fájlt szeretnénk szerkeszteni, akkor a ***Vim*** segítségével most a következőképpen kell megnyitnunk:

```
$ vim common/backend/
➥networking/tcp.c
```

Régen, a *ctags* nélkül a következőt tettük volna:

```
$ cd common/backend/networking
$ vim tcp.c
```

Mivel a *tags* fájl a hierarchiában a *common* könyvtár fölött helyezkedik el, a *Vim* automatikusan tudni fogja ennek a helyét.
Van ugyanakkor egy másik módszer is. Megtehetjük, hogy a fent említett második (hagyományos) módszerrel nyitjuk meg a kérdéses fájl, majd már a *Vim*-ben kiadjuk a következő parancsot:

```
:se tags=../../../tags
```

Szerintem amúgy az első módszer könnyebben használható. Ha bármilyen módon megnyitottuk a fájlunkat, a függvények definíciói között a *Ctrl-]* billentyűkombinációval mozoghatunk. Ha oda akarunk ugrani bármilyen nyelvi szerkezet (függvény, konstans, makró vagy bármi más) definíciójára, vigyük fölé a kurzort, majd nyomjuk meg a *Ctrl-]* billentyűket. Röviden tehát egy függvény definíciójának megkereséséhez előbb meg kell találnunk egy pontot, ahol meghívjuk azt. Az már teljesen lényegtelen, hogy a függvény definíciója az adott fájlban, vagy egy egészen más könyvtárban és fájlban van. Tegyük fel például, hogy a *tcp.c* fájlban meghívunk egy `drawscreen()` nevű függvényt.
A tagek segítségével a *Vim* még akkor is megtalálja a függvény meghatározását, ha az azt tartalmazó fájl történetesen a *common/gui*-ban van.
Ha a keresés után vissza akarunk ugrani oda, ahonnan jöttünk, nyomjuk meg a *Ctrl-T* billentyűkombinációt. Újabb kereséshez nyomjuk meg újra a *Ctrl-]* billentyűket, ami után persze megint a *Ctrl-T*-vel térhetünk majd vissza.
Ha a keresett dolog nevének csak egy részét ismerjük, akkor a következőképpen is eljárhatunk:

```
:ta /function
```

Ha több ilyen rész is van a forráskódban, ez a parancs az első elforduláshoz ugrik. A következőre a `:tn` paranccsal válthatunk.
Ha egy függvénynek több definíciója is van, és választani szeretnénk közülük, megnyomhatjuk a *G Ctrl-]* billentyűket, vagy használhatjuk a `:tselect <tagname>` parancsot. Ezzel a módszerrel anélkül mozoghatunk a forrásban az egyes függvények között hogy tudnunk kellene, melyik pontosan hol található.

A Vim és a Cscope

A *cscope* egy másik hatékony navigációs eszköz. Íme a program által felkínált menü:

```
Find this C symbol:
Find this global definition:
Find functions called by this
➥function:
Find functions calling this
➥function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this
➥file:
```

A programmal a *Vim* is képes együttműködni, így könnyedén ötvözhetjük az egyik hatékonyságát a másik kényelmével. A kapcsolat megteremtéséhez az egyetlen dolog, amit tennünk kell a

```
:cs add cscope.out.
```

parancs kiadása.
Akárcsak az előbb tárgyalt *ctags*, a *cscope* is egy indexfájlt hoz létre *cscope.out* néven. Ezt a következő paranccsal állíthatjuk elő:

```
$ cscope -Rbq
```

A szabályok is ugyanazok, mint a *ctags* esetében, vagyis ha a forráskód több könyvtárban szétszórtan helyezkedik el, akkor a fenti parancsot a könyvtárszerkezet gyökerében kell kiadnunk.
A történet folytatása is hasonló.
Ha egy a könyvtárszerkezet belsejében található fájlt nyitunk meg, akkor is a gyökérből kell indulnunk, majd a *Vim*-ben ki kell adni a `:cs add` *cscope.out* parancsot a kapcsolat felépítéséhez. A létező *cscope* kapcsolatokat a `:cs show` paranccsal listázhatjuk.
Hogy mi mindenre kereshetünk a *Vim*-en belülről, azt a `:cs <CR>` paranccsal nézhetjük meg. Ha például bele akarunk nézni egy forrásba, vagy egy fejlécállományba, egyszerűen gépeljük be a `:cs f f stdio.h` vagy a `:cs f f foo.c` parancsot.
Ha a *foo.c*-ben meghívott függvényekre akarunk rákeresi, használjuk a `:cs f d foo.c` parancsot. Ez megjeleníti az összes innen hívott függvény nevét. Ha fordított keresést szeretnénk végezni, vagyis azoknak a függvényeknek a nevére vagyunk kíváncsiak, amelyek a *foo.c*-re hivatkoznak, a `:cs f c foo.c` utasítást használjuk. Kereshetünk egrep minta alapján is a következőképpen: :cs f e varName és így tovább. A lehetőségeket a magában kiadott :cs parancs segítségével listázhatjuk.
Ha sikeresen beüzemeltük a *ctags* és a *cscope* rendszert is, kombinálhatjuk is őket, például így: `:cstag /foo`. Ez a parancs megkeresi az összes olyan függvényt, felsorolt típust vagy bármi egyebet, ami tartalmazza a `foo` karakterláncot.

A Vim és a szintaktikai kiemelés

Ha olyan szolgáltatása a *Vim*-nek, ami fölébe helyezi bármely más szerkesztőnek vagy integrált fejlesztőkörnyezetnek (*IDE*), nos akkor ez a szintaktikai kiemelés. A *Vim* színező képessége egyszerűen élvezetté teszi a vele való munkát. A szöveg szintaktika szerinti kiszínezése természetesen nem csak az életünket teszi színesebbé, hanem jelentsen megkönnyíti az elgépelések és egyéb hibák megtalálását akár a fordítás előtt. A színezés változása révén például azonnal kiszúrhatjuk a szinte leggyakoribb programozási hibát, a zárójelek hiányát. Azonnal látni lehet, ha nem zártunk be egy idézőjelet, vagy ha egy megjegyzés végéről hiányzik a `*/` jel. Szintén rögtön kibukik, ha netán megjegyzéseket akarunk egymásba ágyazni. Egy *C* programozó számára mindez nagyon hasznos, sőt néha életmentő lehet.
A *Vim* szintaktika szerinti színezőképességének bekapcsolásához általában semmit nem kell tennünk. Ha azonban valamiért úgy települne fel a program, hogy ez a szolgáltatása nincs alapértelmezésként bekapcsolva, akkor a `:sy on` paranccsal gondoskodhatunk a helyes működésről.

Akárcsak bármely más parancsot, természetesen ezt is felvehetjük a *~/.vimrc* fájlunkba.
Ha a szöveg még mindig nem színes, akkor a terminál beállításaival van valami gond. Ilyenkor azért még próbáljuk ki a `:se filetype` és a `:sy eanble` parancsokat.
Tegyük fel, hogy sikerült elővarázsolnunk a színeket, de valamelyik közülük nincs ínyünkre. A kék például elég rosszul látszik sötét háttér előtt, márpedig a *C* forráskódok megjegyzései alapértelmezésként ezzel a színnel jelennek meg. Ilyenkor több lehetőségünk is van. Használhatjuk például a `:se background=dark` parancsot, amivel a program tudtára adjuk, hogy milyen színű a háttér. De az is megoldás, ha kikapcsoljuk a megjegyzések színezését a `:highlight clear comment` paranccsal.
Aztán ha ez még mindig nem elég, akkor meg is változtathatjuk az alapértelmezett színeket. Ehhez először adjuk ki a `:syntax` parancsot, ami megjeleníti az összes szintaktikai elemet. Ezután adjuk meg a megváltoztatni kívánt csoport nevét. Ha például azt akarjuk, hogy a karakterláncok ragyogó fehérrel jelenjenek meg, ami fekete háttér előtt nyilván kiválóan olvasható, egyszerűen gépeljük be a következő parancsot:

```
:highlight String ctermfg=white
```

Aki *gvim*-et használ, annak viszont a következőképpen kell eljárnia:

```
:highlight String guifg=white
```

Természetesen bármely más szintaktikai csoport színét megváltoztathatjuk. A leggyakoribb elemek a következők: `Statement`, `Label`, `Conditional`, `Repeat`, `Todo` és `Special`. A színen kívül megváltoztathatjuk a kiemelés módját is. Beállíthatjuk például, hogy egy elemtípus legyen alahúzva, vagy legyen félkövér. Ha például a `NOTE`, `FIXME`, `TODO` vagy `XXX` elemeket alahúzva szeretnénk megjeleníteni, a következőt kell tennünk:

```
:highlight Todo cterm=underline
```

Így érhetjük el, hogy a színezéssel egyidejűleg az adott elem félkövér is legyen:

```
:highlight Repeat
➥ctermfg=yellow cterm=bold
```

Ezekkel a parancsokkal létrehozhatunk akár egy teljes kiemelési szabályrendszert, amit természetesen megint a *~/.vimrc* fájlban tárolhatunk.
Így valahányszor elindítjuk a *Vim*-et, a beállításokat végző parancsok rögtön az elején automatikusan lefutnak, és munka közben élvezhetjük kedvenc színeinket.

Változónevek automatikus kiegészítése

A *Vim* arra is képes, hogy automatikusan kiegészítse a változók neveit. Gépelés közben a ***Ctrl-N*** vagy a ***Ctrl-P*** segítségével válthatunk beszúrás üzemmódba. Fontos megjegyezni, hogy a kiegészítés csak ebben az üzemmódban működik. Minden más, eddig említett parancsot parancs üzemmódban kell használnunk. Ha több lehetséges kiegészítés is van, akkor a ***Ctrl-N*** többszöri megnyomásával léptethetjük a lehetőségeket.
Ennek a szolgáltatásnak az a legfőbb haszna, hogy segítségével könnyebb elkerülni az elgépeléseket, hiszen a szöveg jelentős részét nem mi visszük be kézzel, hanem a program „emlékezetből". A névkiegészítés akkor működik a leghatékonyabban, ha engedélyeztük a *Vim*-nek a *ctags* fájl használatát.

A forráskód formázása

A *Vim* elég jól érti a *C* nyelvet ahhoz, hogy automatikusan kezelni tudja a beljebbezést. Alapértelmezésként ehhez tabulátorokat használ, ami egyes programozókból ellenérzéseket válthat ki. Ha a teljes forráskódból el száműzni szeretnénk a tabulátorokat, adjuk ki a következő parancsot:

```
:set expandtab
:retab
```

Ez valamennyi tabulátort szóközökké alakít úgy, hogy a formázás változatlan maradjon. Miközben gépeljük a kódot, a *Vim* automatikusan formázza azt. Ez nyilván megkönnyíti az egymáshoz tartozó zárójelek felderítését. Ehhez amúgy használhatjuk a % parancsot is parancs üzemmódban. Álljunk rá a megfelelő zárójelre – legyen az kerek, szögletes vagy kapcsos – majd nyomjuk meg a % billentyűt. Erre a *Vim* odaugrik a megfelelő nyitó vagy záró zárójelre. Ugyanez a módszer működik a megjegyzésekre, valamint az `#if`, `#ifdef` és `#endif` szerkezetekre is.
Ha befejeztük a kód begépelését, és egyetlen paranccsal akarjuk megformázni, váltsunk parancsmódba, és gépeljük be a következőt: `gg=G`. Ezután szükség szerint átalakíthatjuk a tabulátorokat szóközökké a fentebb említett módszerrel. A megjegyzések formázását a gq parancs végzi. Ha csak egy kódrészletet akarunk beljebb tolni, jelöljük ki, és használjuk az = parancsot.
Végül ha valakit netán kifejezetten bosszantana a *Vim* automatikus formázása, ki is kapcsolhatja az a `:se nocindent` paranccsal. Érdemes ugyanakkor megemlíteni, hogy a formázásra más lehetőségek is vannak, amelyekről a `:help indent.txt` paranccsal olvashatunk.

A súgó

A *Vim* igen terjedelmes dokumentációval rendelkezik, amit a `:help` paranccsal böngészhetünk. Ha egy konkrét téma leírására akarunk ugrani, álljunk a megfelelő türkiz színű szövegrészre, majd üssük le a ***Ctrl-]*** billentyűkombinációt. A *Vim* súgója egyébként ugyanazt a navigációs mechanizmust használja, mint amit a tag-eknél láttunk.

Linux Journal 2005. 140 szám

Girish Venkatachalam
Szeret nyílt forrású operációs rendszerekkel, például OpenBSD-vel, FreeBSD-vel no és Debian GNU/Linuxszal játszadozni. Amikor nem a hackeléssel van elfoglalva, szeret biciklizni is. A girish1729@gmail.com érhető el.

KAPCSOLÓDÓ CÍMEK

A cikkhez tartozó elektronikus források a következő helyen találhatók:
➲ www.linuxjournal.com/article/8455