Feladat-automatizálás az Aap segítségével

Az Aap rugalmas eszköz, amely képes mindenre, amire a make, sőt még egyébre is. Most megtudhatjuk, hogyan hozhatunk létre mobil megoldásokat a weboldalunk kezeléséhez és a programjaink lefordításához.

Sokan Makefile-t, vagy héjprogramokat használnak a műveletek önműködővé tételéhez, ha egy fájl változása valamilyen művelet végrehajtását teszi szükségessé. Elvégezzük a fájlon a kívánt módosításokat, majd segítségül hívjuk a `make`-et remélve, hogy a változtatásokkal kapcsolatos minden szükséges teendőt elvégeztük. Gyakran előfordul, hogy a Makefile-lal kapcsolatban cselhez kell folyamodnunk, és sokszor jutunk odáig, hogy a `touch` segítségével kell valamilyen hiányzó függőséget pótolnunk. Ez oda vezethet, hogy amikor mások kezébe kerül a gondosan beállított Makefile-unk, nehezen tudják kibogozni, hogyan is működik.

Az ilyen feladatok sokkal könnyebben és megbízhatóbban végezhetők el az ***Aap*** segítségével, mint a `make`-kel, sőt előbbi rendelkezik beépített Internet-támogatással is.

A szükséges letöltések és feltöltések végrehajtódnak anélkül, hogy erre parancsokat kellene megadnunk, vagy időbélyegző-fájlokat használnánk. A megbízhatóság forrása az önműködő függőség-felismerés és az aláírások használata az időbélyegzők helyett. Az Aap segítségével könnyebben adhatjuk meg az elvégzendő feladatot és így kevesebb hibát vétünk. Szükség esetén használhatjuk a héj parancsait is.

Ebben a cikkben két példán keresztül világítjuk meg az Aap működését: az első egy weboldal kezelése, a második pedig egy program lefordítása.

Az Aap telepítése

Az `Aap` használatához szükségünk van a Python 1.5-ös vagy újabb változatára. Ha abban a valószínűtlen helyzetben vagyunk, hogy nincs Python a rendszerünkön, töltsük le a ➲ http:\\www.Python.org oldalról, telepítsük a Linux rendszerünk telepítőlemezéről, vagy frissítsük a rendszerünket. Az `Aap` telepítése négy egyszerű lépésben elvégezhető. Kezdjük a legfrissebb ***Aap.zip*** csomag letöltésével (lásd a hálózatos ***Resources*** – források részt), ezután bontsuk ki a csomagot egy ideiglenes könyvtárba (`unzip aap-1.53.zip`).

Ha root-ként jelentkeztünk be, futtassuk az `./aap install` programot. Ha közönséges felhasználók vagyunk, telepítsük a programot a saját könyvtárunkba az `./aap install PREFIX=$HOME` paranccsal. Az `Aap` egy GNU GPL licenc alatt terjesztett nyílt forráskódú program.

Egy weboldal karbantartása

A friss `Aap` programmal felvértezve vágjunk bele az ismerkedésbe, és egy példafeladaton keresztül vizsgáljuk meg a program használatát. Korábban létrehoztunk egy egyszerű weboldalt HTML fájlokkal és képekkel. A fájlok jelenleg a helyi gépünkön vannak, fel kellene tölteni azokat a webkiszolgálóra. Az *1. listán* láthatjuk azt az Aap-receptet, amely elvégzi ezt a feladatot. Receptnek az `Aap` parancsfájljait nevezzük.

Mentsük ezt a parancsfájlt ***main.aap*** néven. Ez az a fájl, aminek a létrehozása a Makefile feladata: az alapértelmezett esetben futtatandó fájl. Az `aap` paraméterek nélküli futtatása az aktuális könyvtárban lévő ***main.aap*** fájl végrehajtását eredményezi.

Egy Aap-receptben lévő megjegyzések kettős kereszttel (`#`) kezdődnek és a sor végéig tartanak, éppen úgy, mint egy `Makefile` vagy héjprogram esetén. A recept első végrehajtandó sora egy értékadás: a `Files` változóhoz hozzárendelődik a feltöltendő fájlok listája. Nincsenek perjelek vagy az értékadás végét jelző egyéb írásjelek, az `Aap` a parancs folytatását a használt behúzás mértékéből ismeri fel. Ez első pillantásra furcsa lehet, de hamar hozzá lehet szokni. Ezzel az eljárással kiküszöbölhetjük az írásjelek használatából adódó hibalehetőséget és kényszerítjük magunkat a könnyen olvasható külalak használatára. A sorok behúzására használhatjuk a szóközt és a TAB billentyűt egyaránt.

Az `:attr` kulcsszóval kezdődő sor egy `Aap`-parancs. Minden Aap-parancs kettősponttal kezdődik, így könnyen felismerhető. Ez a parancs a `publish` tulajdonságot rendeli a megadott paramétereihez, ez határozza meg a fájlok feltöltésének a helyét, amikor azok közzétételre kerülnek. Az itt használt eljárás az ***scp:***//, a secure copy (biztonságos másolás). Az ***rsync:***// és az ***ftp:***// a másik két támogatott eljárás. Az `:attr` parancs utolsó paramétere a `$Files`, a Files változó értéke. A tulajdonság a `$Files` minden egyes eleméhez hozzárendelődik.

Amikor az `Aap`-t paraméter nélkül futtatjuk, az `all` célobjektum által meghatározott elemeket frissíti. Az utolsó sor azt határozza meg, hogy az alapértelmezett all célobjektum a publish tulajdonságtól függjön. Ez egy speciális célobjektum, amely arra utasítja az `Aap`-t, hogy azokat a tételeket töltse fel, amelyek publish tulajdonsággal rendelkeznek.

Most már szerkeszthetjük a HTML-fájljainkat, képeket adhatunk hozzájuk és nézegethetjük az eredményt a saját gépünkön. Amikor elégedettek vagyunk az eredménnyel, futtathatjuk az `aap`-t. Az `Aap` észleli, hogy mely fájlok változtak meg és feltölti azokat. Ehhez aláírásokat (ellenőrző összegeket) használ, így ha egy fájl korábbi változatát állítjuk vissza, akkor is megfelelően működik. Ha a `make`-et használnánk, a visszaállított fájlhoz is hozzá kellene nyúlnunk, hogy beállítsuk az időbélyegzőjét.
Ha ki szeretnénk próbálni ezt a példát, de nincs olyan kiszolgálónk, ahova feltölthetnénk az anyagot, használhatjuk a `file:/tmp/html/%file%` kifejezést a publish tulajdonság beállításánál. Ebben az esetben az `aap` szükség esetén létrehozza a */tmp/html* könyvtárt. Egy figyelmeztetés: az `Aap` nem törli a kiszolgálóról a már nem használt fájlokat. Ezt a `make` sem teszi meg nekünk. Saját magunknak kell a törlést kézzel elvégeznünk. Remélhetőleg az Aap rövid időn belül kiegészül az önműködő törlés lehetőségével is.

A képfájlok listázása

A képek kiválasztásánál használtunk egy dzsókerkaraktert: ***images/*.png***. Ez ugyan kényelmes, de magában rejti azt a veszélyt, hogy olyan képek is felkerülnek, amelyeket nem akarunk feltölteni. Minden egyes fájl megnevezésével elkerülhető ez a csapda, de így a felsorolásból könnyen kifelejthetünk valamit. Mivel ez egy elég általános probléma, az Aap egy függvényt biztosít a képek fájlneveinek a HTML-fájlokból történő kiszűrésére. A *2. lista* mutatja ennek a módját. A `get_html_images` Python-függvényt hívjuk segítségül, az aposztrofok között egy Python-kifejezés szerepel. Az Aap kiértékeli a kifejezést, majd az eredményt, a képfájlok neveit, berakja a helyükre.
A `get_html_images()` függvénynek vannak azonban korlátai is: csak tiszta HTML-fájlokkal képes dolgozni, amelyekben a képek relatív elérési útvonallal rendelkeznek.

HTML-fájlok előállítása

A legtöbb HTML-fájl fejrészből, címből, törzsrészből és lábrészből áll. Nyilvánvaló, hogy nem akarjuk minden egyes alkalommal újra begépelni a közös részeket. Egyszerű megoldás a több fájlból történő összefűzés. A *3. lista* az ezt megvalósító receptet mutatja. Öt részt használunk: `header` (fejrész), `title` (cím), `middle` (középrész), `contents` (tartalom), és `footer` (lábrész). A cím és a tartalom minden oldalon különböznek, de a másik három rész megegyezik.
A *2.* és *3. lista* közti lényeges eltérés a *3. listában* hozzáadott `:rule` parancs. Ez azt meghatározza, hogy a célobjektum (a HTML-fájl) öt forrásfájltól függ, és a parancs ezeket sorolja fel, aminek alapján a forrásfájlokból előáll a célfájl. A % jel a fájl neve helyett használatos, hasonlóan a * dzsókerkarakterhez. A `rule` parancs minden %-jele ugyanazt a nevet helyettesíti, így az ***index.html*** előállításakor az index szó helyett használatos. A forrásfájlok közt foglal helyet tehát az ***index_title.part*** és az ***index.part*** fájl. A `:rule` sora alatt azoknak a parancsoknak a – behúzott formában lévő – felsorolása következik, amelyek akkor futnak le, amikor a rule parancs célpontjának frissítése szükségessé válik. Vagyis a recept két szintre bomlik: a felső szinten lévő parancsok akkor futnak le, amikor a recept olvasásra kerül, a `rule`-parancsblokk pedig később, amikor szükségessé válik.

1. lista Recept fájlok webkiszolgálóra való feltöltésére

```
# A feltöltendő fájlok listája.
Files = index.html
        info.html
        download.html
        images/*.png

# A publish tulajdonság jelzi a feltöltés
# helyét.
:attr {publish =
➥scp://my.server.net/html/%file%}
        $Files

# Amikor target (cél) meghatározása nélkül
# futtatjuk: nyilvánossá teszi a fájlokat.
all : publish
```

2. lista A képek fájlneveinek kinyerése a HTML-fájlból

```
# A feltöltendő fájlok listája.
Files = index.html
        info.html
        download.html
Files += `get_html_images(Files)`

# A publish tulajdonság jelzi a feltöltés
# helyét.
:attr {publish =
➥scp://my.server.net/html/%file%}
        $Files

# Amikor target (cél) meghatározása nélkül
# futtatjuk: nyilvánossá teszi a fájlokat.
all : publish
```

A `:cat` parancs összefűzi a fájlokat, hasonlóan a UNIX `cat` parancsához. Valójában ennél sokkal többre képes, például egy megadott címről történő fájlbeolvasásra. A `rule` parancsban a `$source` a forrásfájlok teljes listáját jelenti.
A HTML-fájlokat létre kell hoznunk, még mielőtt kinyernénk a bennük szereplő képfájlok listáját. Ehhez az `:update` parancsot kell segítségül hívnunk a `get_html_images()` függvény hívását megelőzően. Ennek hatására a megadott szabály alapján megtörténik a HTML-fájlok frissítése. Ez a parancs a recept felső szintjén foglal helyet, vagyis mindig végrehajtódik, amikor az `Aap` olvassa a receptet.
Most, hogy ennyi fájlunk van, honnan fogja az `Aap` tudni, hogy milyen tennivalókat kell végrehajtania? Az `Aap` erre a függőségeket használja, akárcsak a `make`. Azzal a célobjektummal kezdi, amit a parancssorban megadunk, ha nem adunk meg semmit, akkor az `all` (minden) a feltételezett cél. Az `Aap` ezután megkeresi azokat a függőségeket és szabályokat, amelyekben ez a cél megjelenik a kettőspont előtt. A kettőspont lényegében a függést jelenti, a kettőspont

3. lista HTML fájl létrehozása öt részből

```
Files = index.html
        info.html
        download.html

:rule %.html : header.part
               %_title.part
               middle.part
               %.part
               footer.part
    :cat $source >! $target

:update $Files
Files += `get_html_images(Files)`

:attr {publish =
➥scp://my.server.net/html/%file%}
        $Files

all : publish
```

után helyezkednek el azok a forrásfájlok, amelyektől a cél függ. Ezután ezeket a forrásfájlokat vizsgálja meg az Aap és megkeresi az összes olyan szabályt, amelyben ezek célként szerepelnek. A folyamat rekurzívan folytatódik mindaddig, amíg el nem fogynak a szabályok. Az eredmény egy függőségekből álló faszerkezet. Az `Aap` ezután a fa végétől (a legmélyebb résztől) kezdve lefuttatja azokat a parancsokat, amelyek ennek a függőségi rendszernek a felépítéséhez szükségesek. Kicsit bonyolultnak tűnik, igaz? Mivel azonban az `Aap` elvégzi ezeket a teendőket, nekünk nincs más dolgunk, mint hogy minden egyes célnál megadjuk, hogy milyen forrásoktól áll függésben. Az `Aap` ezután már el tudja dönteni, mi a teendő.

Időbélyegző létrehozása

Egy hasznos kiegészítésként adjunk időbélyegzőt is a HTML-fájlhoz, így a weboldalon láthatóvá válik, hogy mikor volt az oldal legutoljára frissítve. Írjuk be valahova a fájl lábrészébe (***footer.part***) a `@TIMESTAMP@` karakterláncot.
A *4. lista* mutatja azt a szabályt, amelynek alapján ez a karakterlánc az aktuális dátummal lesz helyettesítve. A recept többi része ugyanaz, mint a *3. listában*. Az `:eval` parancs kiértékeli a Python-kifejezést, amelyben a `string.replace` egy szabályos Python-függvény egy karakterlánc másikkal való kicserélésére. Ezzel a módszerrel bármilyen Python-kifejezést használhatunk a szöveg szűrésére. A HTML-oldal épp úgy halad át az `:eval` parancs csővezetékén, mintha a héj alól tennénk ugyanezt. A szabály használatának első alkalmával minden HTML-fájl frissítésre kerül. Ennek oka, hogy az Aap megjegyzi a parancsok aláírását, emiatt egy receptben történt változtatás után nem kell foglalkoznunk a fájlok újbóli előállításának végrehajtatásával.

Feltöltés az rsync segítségével

Amikor egy honlapon valamilyen kis módosítást eszközölünk, nem túl gazdaságos teljes fájlt feltölteni.

4. lista Szabály a létrehozott HTML-fájl időbélyegzővel történő ellátására

```
:rule %.html : header.part
               %_title.part
               middle.part
               %.part
               footer.part
    :print Generating $-target
    :cat $source
        | :eval string.replace(stdin,
                  ➥´@TIMESTAMP@´, _no.DATESTR)
        >! $target
```

A feltöltés hatékony módszere az `rsync` segítségével valósítható meg, amely a fájlnak csak azokat a részeit tölti fel, amelyek megváltoztak. Az Aap akkor alkalmazza az `rsync`-et, ha a `publish` tulajdonságban megtalálja az `rsync://` beállítást. Alapértelmezésben az `rsync` SSH-kapcsolaton keresztül kommunikál, ezt a beállítást a `$RSYNC` változó megváltoztatásával módosíthatjuk. Az `rsync` nem szabványos parancs, amennyiben nincs a rendszerünkre telepítve, részünk lehet az Aap egy hasznos szolgáltatásában: a program választási lehetőségként felajánlja az `rsync` telepítését.

```
% aap
Aap: Uploading [´index.html´] to

rsync://my.server.net/html/index.html
Cannot find package "rsync"!
1. Let Aap attempt installing the package
2. Retry (install it yourself first)
q. Quit
Choice:
```

Az `Aap` rendelkezik a programtelepítés képességével, éss ha szükséges, az `Aap` honlapjáról még recept letöltésére is képes, amely leírja az adott program telepítésének módját. Itt kapóra jön az Aap letöltési képessége.

Programfordítás az Aap-vel

Az `Aap` támogatja a C vagy C++ forráskódból történő fordítást is. Íme egy egysoros recept, amely lefordítja a négy C forrásfájlból álló `myprog` nevű programot:

```
:program myprog : main.c common.c various.c args.c
```

A recept egyszerűsége ellenére az Aap gondoskodik az alábbiakról:

- A függőségeket önműködően feltérképezi. Nincs szükség a befoglalandó fejállomány megadására vagy a `make depend` parancs futtatására.
- A recept a legtöbb rendszeren módosítás nélkül működik. Az `Aap` megkeresi a használandó fordító és összefűző programot és meghatározza a szükséges paramétereket.
- Az objektumfájlok minden egyes rendszer esetén külön könyvtárban tárolódnak, így több változatot is fordíthatunk anélkül, hogy az előzőt el kellene távolítanunk.

- Az Aap létrehoz egy naplófájlt, az ***AAPDIR/log*** fájlt, amely a végrehajtott műveleteket részletezi. Ha a fordításunk nem sikerül és a kimenet már nem látszik a képernyőn, nincs szükség arra, hogy a kimenet átirányításával újrafuttassuk a fordítást.
- Néhány alapértelmezett cél önműködően létrejön: az `aap install` telepíti a programot, míg az `aap clean` törli a létrehozott fájlokat.

Ugyanezt a munkát a make-kel is elvégezhetnénk, de a Makefile sokkal hosszabb lenne.

Változatok létrehozása a fordításkor

Most hozzuk létre egy program két változatát, egy kiadásra szánt verziót, egyet pedig hibakeresés céljából. Az Aap támogatja a különféle változatok létrehozását. Mindössze azt kell megadnunk, hogy milyen változatokra lesz szükségünk, és mi különbözteti meg ezeket egymástól. Az 5. lista mutatja a szóban forgó receptet. Az első sor, a `:variant` parancs meghatározza a változat nevét, amelyre a fordításkor hivatkozhatunk. Ezt a változót parancssorban állíthatjuk be; az `aap Build=debug` a hibakereső változatot adja. Ha nem adunk meg paramétert, akkor a nyilvános változat készül el, mivel az szerepel először. A behúzás mértéke azonosítja a `:variant` parancs további részeit. A lehetséges értékek behúzása kicsit kisebb, az egyes értékeknél használt parancsok nagyobb behúzással szerepelnek. Az egyes részek illesztése kötelező, ezzel az olvasás is könnyebbé válik. A `release` változat beállítja az `OPTIMIZE` változót. Ez egy nullától kilencig terjedő szám lehet, ami az optimalizálás szintjét határozza meg. Ez önműködően beállítja a használt fordítóprogram megfelelő paraméterét. A `debug` változat a `DEBUG` változót `yes` értékre állítja. Az alapértelmezett érték `no`. A `Target` változó az eredményként kapott program nevét tartalmazza. A két változat különböző nevet használ, így mindkét program létezhet egy időben. A változatok ilyen módon való használatának nagy előnye, hogy az objektumfájlok minden változathoz önműködően külön könyvtárban tárolódnak. Amikor átváltunk a két változat között, nem szabad elfelejtenünk, hogy az Aap nem hoz létre újra minden fájlt.

Fordítás más nyelvből

Ha nem C vagy C++ nyelvű kódot szeretnénk fordítani, szükség van a megfelelő nyelvmodul importálására. Néhány alapmodult már az Aap is magában foglal. Például a következőképpen fordíthatunk le D nyelvű forráskódot (a D egy új programozási nyelv):

```
:import d
:program myprog : main.d common.d various.d args.d
```

Az `:import d` parancs a D nyelv támogatásának betöltésére szolgál. Ettől eltekintve a folyamat hasonló a C-források fordításához. Saját magunk is írhatunk olyan modult, amely megteremti egy nyelv támogatottságát. Mivel az Aap nyílt forráskódú alkalmazás, nyugodtan átadhatjuk a modult az Aap részeként való terjesztésre. Amíg ez megtörténik, helyezzük el a fájlt az Aap modulkönyvtárába, amely bővítményként működik.

5. lista Kibocsátársa és hibakeresésre szánt változatok fordítása

```
:variant Build
   release
      OPTIMIZE = 4
      Target = myprog
   debug
      DEBUG = yes
      Target = myprogd

:program $Target : main.c common.c various.c
➥args.c
```

6. Lista Az Aap használata a make helyettesítésére.

```
 all : hello

# A hello program manuális fordítása
hello : hello.c
   :sys cc -o $target $source

# Egy C program előállításának nehézkes módja.
hello.c:
   :print Generating $target
   :print >! $target $(#)include $(<)stdio.h$(>)
   :print >> $target main() {
   :print >> $target   printf("Hello World!\n");
   :print >> $target   return 0;
   :print >> $target }
```

Egy KDE-alkalmazás fordítása

Egy KDE-alkalmazás lefordítása egy csomó eszköz használatát vonja maga után, például a Qt Designer alkalmazását a párbeszédablakok létrehozására, a fejállományok létrehozását a felhasználói felület leírásaiból és a folyamatközi kommunikáció kódjának előállítását. Mindezek ellenére egy KDE-program lefordítását végző recept is lehet ilyen egyszerű:

```
:import kde
:program logger : main.cpp
                  logwidget.ui
                  dcop.h {filetype = skel}
                         {var_OBJSUF = _skel.o}
```

A három bemeneti fájl közül a `main.cpp` közvetlenül lefordítható. A Qt Designer ***logwidget.ui*** nevű fájlját először az `uic` használatával kell feldolgozni, melynek során létrejön egy `include`-fájl, ezután pedig a `moc`-ot kell használni. Az Aap felismeri az ***.ui*** kiterjesztést és odafigyel minderre. Az ilyen többlépcsős fordítófolyamat-kezelés, amely során először az `ui`-fájlból `h`-, majd ebből `moc`- és objektumfájl jön létre, az Aap rendkívül hasznos szolgáltatása. Ugyanennek a `Makefile` segítségével történő végrehajtása sokkal több explicit szabályt igényel. A ***dcop.h*** fájl különleges KDE-elemeket tartalmaz, de normál kiterjesztéssel rendelkezik,

7. lista A Python használata a fájlnévlisa előállítására.

```
LASTPATCH = 144

# A foltok fájlneveinek előállítása.
@Patches = ''
@for i in range(1, int(LASTPATCH) + 1):
@    Patches = Patches + ("6.2.%03d " % i)

# Alapértelmezett target (cél): minden foltot
# alkalmazunk.
all: done/$*Patches

# A két könyvtár létezésének biztosítása..
:mkdir {force} patches done

# Szabály a folt alkalmazására.
:rule done/% : patches/% {fetch =
➥ftp://ftp.vim.org/pub/vim/%file%}

    :sys patch < $source
    :touch $target
```

ezért önműködően nem ismerhető fel. Ez az oka, hogy a fájltípus tulajdonságát be kellett állítani. A `:program` parancsnak szintén tudnia kell az objektumfájl nevét, ezt adja meg a `var_OBJSUF` tulajdonság.
A használt KDE-eszközöket nem kell külön megadnunk, ezt az összetettséget a KDE modulfájl rejti el a szemünk elől. Ez sokkal kevésbé bonyolult, mint az `automake` használata.

Az Aap használata a make helyett

Idáig olyan magas szintű `Aap`-parancsokat használtunk, amelyekkel gyorsan megadhatóak a szükséges teendők. A nem szabványos feladatok végrehajtásához pontosan meg kell határoznunk a függőségeket és a végrehajtandó parancsokat. Ez nagyjából úgy működik, mint egy `Makefile`. A héjparancsok mellett használhatunk hordozható `Aap`-parancsokat is. Ha ez sem elég, a Python parancsfájlok is rendelkezésünkre állnak.
A *6. lista* mutatja, hogy miképpen fest egy ilyen alacsony szintű recept. Ebben minden függőséget pontosan meghatároztunk – az `all` a `hello`-tól függ, a `hello` a ***hello.c*** fájlból fordítódik, a ***hello.c*** pedig az alapoktól jön létre lépésenként. Mivel egy receptben a fordítóparancsok Aap-parancsok, a héjparancsok futtatásához szükség van a `:sys` használatára. A példában szereplő `:sys cc` a C fordítóprogramot futtatja. A héjparancsok használatával tehát csökken a receptek hordozhatósága. A ***hello.c*** fájl `:print` parancsok használatával jön létre.
Az első sor a `>! $target` kifejezést használja, amely felülírja az esetleg már létező `hello.c` fájlt. A felkiáltójel nélkül csak egy hibaüzenetet kapnánk, amely a fájl létezéséről tájékoztatna. Ugyanez a sor tartalmazza a `$(#)` kifejezést is, amely megváltoztatja a # karakter megjegyzés kezdetét jelző különleges jelentését. Ugyanígy a `$(<)` és `$(>)` kifejezésekkel kapjuk meg a `<` és `>` karaktereket az átirányítás helyett.
A `hello.c` fájl létrejön, ha még nem létezne, nincs semmilyen forrásfájl-függőség megadva. A fájl egy másik helyzetben is újragenerálódhat: amennyiben valamelyik `:print` parancs megváltozik, mivel ez megváltoztatja a fordítóparancsok aláírását. Amennyiben ezek a parancsok megváltoznak, az `Aap` tudja, hogy a célfájlt újra létre kell hozni.
A fájlt `Aap`-parancsokkal hozzuk létre, nincs szükség héjparancsok használatára. A receptnek ez a része ezért bármilyen rendszeren használható.
Az `Aap`-parancsok száma azonban korlátozott. Ha további szolgáltatásokra van szükségünk de a hordozhatóságot is meg szeretnénk őrizni, a Python parancsfájlok állnak rendelkezésünkre. Az Aap-receptek minden forgalomvezérlési megoldása a Pythonra épül. A *7. listán* láthatunk egy példát egy receptre, amely a `Vim` foltjait telepíti.
Egy ciklust használunk a foltok fájlnévlistájának előállítására a `vim-6.2.001`-től kezdődően és az utolsó foltszámmal bezárólag, amelyet a `LASTPATCH` változó ad meg. Minden egyes foltot le kell tölteni és telepíteni. A `done/$*Patches` kifejezés `$*` jele az `rc`-stílusú változókiterjesztésre utal, a `done/` a Patches minden eleme elé beszúrásra kerül.

Programcsomag telepítése

Korábban említettük, hogy az `Aap` képes az `rsync` telepítésére, ha az nincs fenn a rendszerünkön. A csomagtelepítő folyamat közvetlenül is hívható. Például az Agide telepítéséhez az `aap --install agide` parancsot használhatjuk. Az Agide az A-A-P GUI IDE, az A-A-P projekt egy másik része, amelyet programok fordítására és hibakeresésre használhatunk a `Vim` és `gdb` segítségével.
Bár még a fejlesztés korai szakaszában van, már most elég jó arra, hogy C programokat fejlesszünk és teszteljünk vele. Számos csomag már most elérhető és az idő múlásával egyre több válik azzá.
A csomagok aktuális listája elérhető a ➲ http://www.a-a-p.org/packages.html oldalon. Az `Aap` maga is telepíthető. A legfrissebb változatra való frissítés az `aap --install aap` paranccsal végezhető el. Ha a rendszerünk rendelkezik csomagkezelővel, valószínűleg jobban járunk, ha azt használjuk.

Összegzés

E cikkből képet kaphattunk arról, hogy milyen feladatokat tehetünk önműködővé az `Aap` segítségével.
A kísérletezés során rengeteg segítséget találhatunk az igen alapos súgóban, amely különböző formátumokban letölthető az `Aap` honlapjáról (lásd a cikkhez kapcsolódó címeket). Ezeken az oldalakon rengeteg olyan dolognak a leírását megtalálhatjuk, amire ebben a cikkben nem tudtunk kitérni, ilyen például a CVS használata a változatkezelésre, az önműködő beállítás és sok más.

Linux Journal 2004. május, 121. szám

Bram Moolenaar az Aap projektvezetője és fő fejlesztője. Leginkább a Vim szerkesztőprogramon végzett fejlesztőmunkájáról ismert. Bram Aap-n végzett munkáját a Stichting NLnet támogatta (www.NLnet.nl). A honlapját a www.Moolenaar.net címen találjuk meg.