GoboLinux – Egy alternatív Linux terjesztés

Manapság nagyon sok Linux-terjesztés létezik, ezért nem is tűnik fel szinte senkinek ez a brazil disztribúció (szerintem méltánytalanul). Az egész rendszernek nagyon érdekes filozófiája van, ezért nem ajánlható azoknak, akik szeretik, ha minden lehetséges beállításhoz van akár szöveges, akár grafikus felhasználói felület és nem kedvelik a szokatlant.

Viszont aki szereti, ha a legtöbb programja forrásból van telepítve (akár optimalizáció, akár egyéni konfiguráció vagy akármi más miatt), viszont nem akar „igazi forrás-alapú" disztribúciót (*Gentoo*, *SourceMage*, *Lunar Linux*, stb.) használni, vagy szeretnek „bütykölni" vagy lázadóbb típusúak és szeretik a szokatlan dolgokat kipróbálni, azoknak a *GoboLinux* ajánlható.

A telepítés

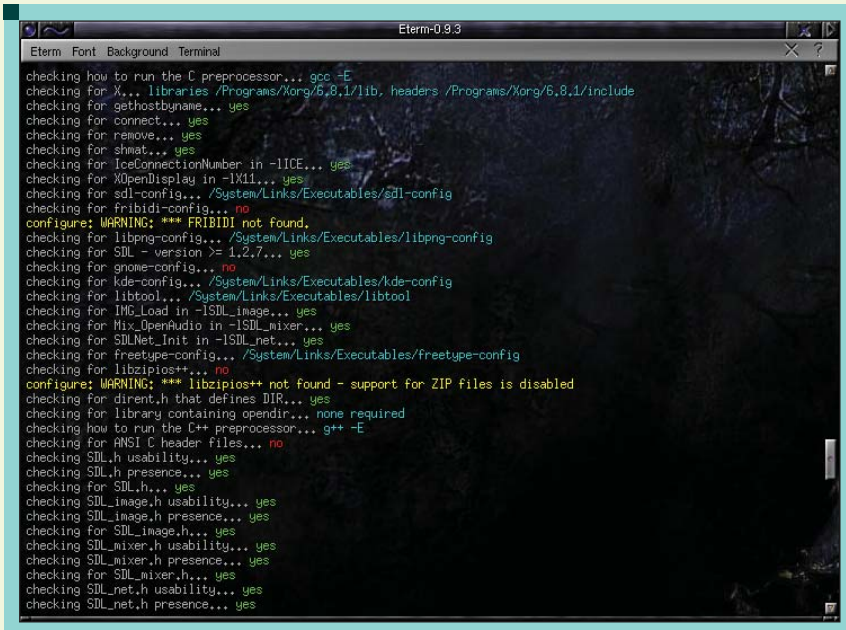
Először a ➲ *www.gobolinux.org* címről töltsük le a megfelelő ISO fájlt, ezt írjuk CD-re és bootoljunk róla. A telepítő-CD egyben *Live-CD*-ként is funkcionál, így vészhelyzetben is jól jöhet. Az indítás végén egy teljes értékű Linux alatt találjuk magunkat, és a `startx` paranccsal egy *KDE*-felületet indíthatunk. Az asztalon találunk egy *Install GoboLinux* ikont, amelyet elindítva a telepítés megkezdődik. Ha nem akarunk (vagy nem tudunk) grafikus felületet használni, akkor a konzolon adjuk ki az `Installer` parancsot (amelyről a rendszer tájékoztat is bennünket).
Maga a telepítés egyértelmű, azaz aki nem először fut neki a dolognak, az biztosan el fog vele boldogulni, így ezt nem is részletezném. Egyetlen egy megjegyzést azért kell tenni: már a bevezetőben is említettem, hogy a *GoboLinuxnak* érdekes filozófiája van. Ezért a *root* felhasználó neve alapértelmezetten nem *root*, hanem *gobo*, viszont ha nagyon ragaszkodunk a hagyományokhoz, nyugodtan lehetünk *root* is.
Amíg a telepítés zajlik, olvassuk tovább a cikket, hogy lássuk, mi az, ami ránk vár illetve miért lehet jobb választás ez a disztribúció, mint a megszokottak.

Bináris disztribúciók és a forrásból való telepítés

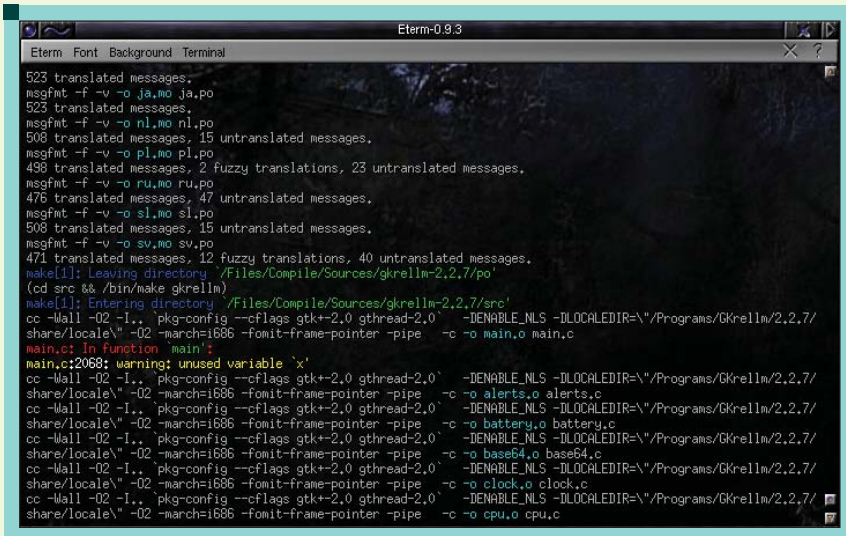
A bináris disztribúciók nagy többségének egyik hátránya, hogy nemigen támogatják, ha forrásból telepít a *root* bármit. Ez a nem-támogatás akkor (is) jelentkezhet, ha egy *X* csomag (egyik) *Y* függősége forráskódból van telepítve, így *Y* nem is jelenik meg a csomagkezelő (*dpkg*, *rpm*, *installpkg*, *pacman*, ...) adatbázisában és *X* telepítésekor hiányolja. Ilyen eset leginkább akkor fordulhat elő, ha egy olyan programot szeretnénk, ami a kedvenc disztribúciónkhoz nem elérhető csomagban, így kénytelenek vagyunk fordítani és a már említett *Y* függőségből frissebb kell neki, mint ami csomagból elérhető. Sokszor a felülírás sem szerencsés, mivel a csomagkezelők egy része a telepített fájlok méretét is tárolja.
Ennek egy lehetséges feloldása, ha a felhasználó a rendszer működéséhez szükséges alapcsomagokat és néhány (vagy sok), viszonylag kevés függőséggel rendelkező programot csomagból telepít, a többit pedig forrásból, tehát a rendszerhez tartozó csomagkezelőt ezután gyakorlatilag nem használja. Ebben az esetben a csomagkezelő előnyeitől is meg kell válnunk, azaz nem fogjuk tudni, miket telepítettünk fel, melyik fájl melyik csomaghoz tartozik, egy csomagban milyen fájlok vannak, sőt, a telepített programok verziószámát a legtöbb esetben nem is tudjuk. Ezen hátrány felismerésével egyidejűleg egy kívánságot fogalmazhatunk meg hallgatólagosan: olyan keretrendszer kell, amely támogatja a forrásból való telepítést és nyilván is tartja a telepített programokat verziószámukkal együtt (a mai modern korban nevetséges lenne mindezt papírral és ceruzával rögzíteni), esetleg a bináris csomagokat is bírja kezelni.

A GoboLinux csomagkezelése

Az alapötlet roppant egyszerű, ennélfogva zseniális: *„a csomagkezelő maga a fájlrendszer"* (idézet a honlapról). Azaz: rakjunk minden programot külön könyvtárba, így egyszerűen tudni fogjuk, ki hova tartozik és tudni fogjuk azt is, mik és milyen verzióval vannak feltelepítve. Egy példával megvilágítva rögtön világos lesz az alapelv: legyen a telepítendő program a *bash*, ennek is a 3.1-s verziója. *GoboLinux* alatt ez azt jelenti, hogy van egy */Programs* könyvtár, itt van az *összes* telepített program. Tehát a *bash* a */Programs/Bash/* könyvtárban.
A beállításai (ami normális esetben a */etc*) a */Programs/Bash/Settings* alkönyvtárban, a kifejezetten 3.1-es



1. ábra PrepareProgram színes kimenete



2. ábra A make színes kimenete

verzióhoz tartozó állományok pedig a */Programs/Bash/3.1/{bin,doc,man,lib,include,...}* könyvtárakban.
Ahhoz, hogy a megszokott módon lehessen futtatni a parancsokat („begépelem, Enter, és fut”), két lehetőség van: vagy az összes telepített programban szereplő ***bin*** (és ***sbin***) könyvtárt felvesszük a PATH környezeti változóba (ami valljuk be, sok telepített program esetén használhatatlan), vagy pedig egy könyvárban létrehozok minden egyes ***bin***-beli állományról egy szimbolikus linket, és ezt a könyvtárat veszem fel a PATH-ba. Nyilván a második út a járható, és egyszerűbb is kezelni. *GoboLinux*-ban is ez történik: minden csomag aktuális verziójában levő ***bin*** könyvtár összes fájljára mutat egy szimbolikus link a */System/Links/Executables* könyvtárban, amire pedig egy-egy szimbolikus lánc mutat */bin* és */usr/bin* néven. Hasonló módon a ***lib***-ek a */System/Links/Libraries*-ba, az ***include***-ok a */System/Links/Headers*-be, stb. linkelődnek. Persze a többi könyvtár (*man*, *share*, ...) hasonlóan és mindegyik */System/Links*-belire mutatnak a tradícionális */usr/{bin,include,lib,man,share,...}* linkek.

A verziókezelés is egy egyszerű ötlet alapján megy: az aktuális verzióra egy szimbolikus link mutat ***Current*** néven, tehát ha a ***bash***-ból a 3.1-es verzió van telepítve, akkor a */Programs/Bash/Current* a 3.1-re mutat.
Az ilyen módszerrel megvalósított csomagkezelés legfőbb előnyei a következők:

- Egy programból egyszerre több verzió is lehet fenn, és mindig az a ***Current***, amelyiket akarjuk.
- Mindenféle bűvös csomagkezelő nélkül tudunk lekérdezéseket végrehajtani az alap-parancsokkal (`ls`, `find`, ...).
- Nincs a csomagkezelőnek a szó hétköznapi értelemében vett adatbázisa különféle egzotikus formátumban, amit esetleg egyszerre csak egyvalaki nyithat meg.
- Minden egyes fájlról könnyen meg tudjuk állapítani, hogy melyik csomagnak a része.
- Nincs különbség, hogy egy program forrásból vagy bináris csomagból került a rendszerbe.

Tehát az alapötlet zseniális, már csak meg is kell valósítani. Azzal a néhány (körülbelül száz) bináris csomaggal nincs is gond, azt az `InstallPackage` parancs megoldja. Mi a helyzet a forrásokkal?

Forrásból való telepítés, csomagok készítése

A legtöbb forrás telepítése a legtöbb (és a hagyományos) esetben a

```
./configure && make && make
➥install
```

hármassal történik (esetleg még utána `ldconfig`, `makewhatis -u` vagy más egyéb). A *./configure* egyik előnye, hogy legtöbbször meg lehet adni neki különféle könyvtárakat, hogy a lefordított szükséges részeket hova szeretnénk installálni (a `--prefix`, `--libdir`, `--mandir`, stb.).

Ezt mi is megoldhatnánk manuálisan minden egyes programnál, ami elég nehézkes és nem az ilyen aprócseprő munkák miatt használunk ***Linuxot***. Ezen (és sok más) problémára a ***GoboLinux*** nyújt erre egy szkript-gyűjteményt, ennek egyik része a `PrepareProgram` és a már említett `InstallPackage` is. Az előbbi szkript megfelelően paraméterezi a `./configure`-t, tehát helyette a `PrepareProgram Bash 3.1`-et kell futtatni (a `configure` egyéb opcióit, például `--enable-gui`, `--enable-shared`, stb. ez után adhatjuk meg, amit a `PrepareProgram` átad a `configure`-nak). Ezután következik egy csoda: a ./configure fut le szintaktikailag színezve (syntax highlight)! Tehát például a ***no***, ***error*** kulcsszavak pirossal, a ***warning***-ok sárgával, stb (*1. ábra*).
Ezután a szokásos `make` következik, ami szintén színesen fut le (mivel a `make` egy alias a `ColorMake` szkriptre), tehát nagyon jól követhető (*2. ábra*), mikor mi a kimeneti állomány, milyen ***warning***-ok és ***error***-ok keletkeznek a fordítás során.
Miután a programot lefordítottuk, fel is kell telepíteni. Ezt (általában) a

```
make install
```

paranccsal megtehetjük. Ezután a szimlinkeket kell létrehozni, amit nagyobb programok esetén hosszadalmas lenne egyesével. Erre megoldás a `SymlinkProgram` nevű szkript, amelynek a használata az eddigi példával:

```
SymlinkProgram Bash 3.1
```

A `SymlinkProgram` nagyon jól paraméterezhető, tehát egyedi esetekben is jól működik. Egy nagy hiányossága van, amit könnyen lehet pótolni a szkript átírásával: a ***libexec*** könyvtárat egyszerűen kihagyja (és a `PrepareProgram` se kezeli). Ha egy programot el akarunk távolítani, akkor a `DisableProgram` használható, amely megszünteti (törli) a linkeket, majd ezután egyszerűen az `rm -r` paranccsal töröljük az egész könyvtárat (itt jegyezném meg, hogy az összes szkript ***bash***-ban íródott, tehát könnyen követhető és módosítható

azok számára is, akik nemigen ismerik a szkript-nyelveket).
Viszont ha nincs `./configure`, akkor kénytelenek vagyunk magunk megadni a telepítési útvonalat vagy egy beállított `PREFIX` (esetleg `prefix`) környezeti változóval futtatva a `make` parancsot (ez a jobbik eset) vagy pedig magát a `Makefile`-t kell kézzel szerkesztenünk (esetleg a sed-et vagy bármilyen szövegszerkesztőt igénybe véve). Ebben az esetben célszerű létrehozni a megfelelő könyvtárszerkezetet a */Program*-ban (mivel néhány program nem hozza létre magának a könyvtárakat és hibát jelez, ha nem tudja a nem létező helyre bemásolni a megfelelő dolgokat), ezt pedig a

```
PrepareProgram -t Bash 3.1
```

paranccsal tehetjük meg. Ebben az esetben a `-t` opció eredményezi azt, hogy csak a könyvtárstruktúrát hozza létre.
A ***GoboLinux*** a csomagokat (alapértelmezetten) ***.tar.bz2*** formátumban tárolja, így könnyen kezelhetőek. Csomagok készítése (az általam eddig használt disztribúciókkal összevetve, és szerintem amúgy is) nagyon egyszerű a ***CreatePackage*** szkripttel, egyszerűen megadjuk, hogy melyik könyvtárból készítsen csomagot, ő pedig mindent megcsinál. Még függőségi listát is készít, amit az `InstallPackage` értelmez, és megszakítja a telepítést (hacsak nem utasítjuk ennek ellenkezőjére), ha valami nem teljesül. A függőségek meglétének ellenőrzése lényegében annyi, hogy a szkript megnézi, hogy az adott program telepítve van-e a megfelelő verziószámmal, tehát a */**Programs**/**Deps**/**depsVerzió*** könyvtár létezik-e.
Hogy ez bináris csomagból vagy forrásból van-e fenn, teljesen mindegy, ellentétben a legtöbb bináris terjesztéssel.
A honlapomon (➲ ***www.udvzsolt.extra.hu***) néhány csomag található ***GoboLinuxra***, valamint a hivatalos honlapon linkeket találunk hivatalos és nem hivatalos csomagokat tartalmazó tárhelyre/tükrökre. Ne számítsunk sok csomagra, hiszen ez a terjesztés inkább forrás alapú, mint bináris.

A könyvtárszerkezet

A ***GoboLinux*** könyvtár szinten is „szakít a ***UNIX***-os tradíciókkal”, tehát más a könyvtárszerkezet, viszont a megszokott szerkezet is elérhető: például a */**dev*** könyvtár a */**System**/**Kernel**/**Devices***-re, a */**etc*** a */**System**/**Settings***-re, a */**proc*** a */**System**/**Kernel**/**Status***-ra egy-egy link. A fő ***home*** könyvár nem a */**home***, hanem a */**Users*** (a ***root***-nak is itt van a ***home*** könyvtára), a szokványos csatolási pontok a */**mnt*** ill. manapság a */**media***-val szemben */**Mount***. Ha a home-könyvtár külön partíción van és másik ***Linux*** alatt is ezt használjuk,

3. ábra Scandisk-téma bootoláshoz

akkor célszerű a */Users*-re egy linket létrehozni */home*-ként, hogy ne legyen kavarodás.

A boot-olás

Mint ahogy *GoboLinux* alatt eddig szinte semmi se a megszokott, a bootszkriptek sem azok. Nem a leginkább elterjedt *SysV init* szkripteket használja. A bootoláshoz szükséges állományok a */Programs/BootScripts/VerzióSzám* alatt találhatóak, míg a szkriptek a */Programs/BootScripts/Setting/BootScripts*-ben. Ami bootolás során történik:
a */Programs/BootScripts/Settings/Boot Scripts/BootUp* nevű szkript lefut, majd a *Console* és esetleg a *Graphic* (az */etc/inittab* szerkesztgetése nem árt!). A bootolás beállításai a */Programs/BootScripts/Settings/Boot Options*-ben vannak. Itt lehet megadni, hogy milyen kernelmodulokat kell betölteni, a hálózati beállítások, a konzol betűkészlete is itt szerepel. És itt is egy érdekesség: a bootoláshoz (nem framebuffer vagy bootsplash alapon, hanem egyszerű konzolon) még témát is megadhatunk, tehát az üzenetek milyen "körítéssel" jelenjenek meg: lehet *AppleII, CheckList, GoboText, GoboText-II, Hat, Progress, Progress-II, Quotes, ScanDisk, Slack* és *SplitScreen* közül választani (igen, a nevek ismerősek, tehát pl. a *Scandisk* téma úgy néz ki, mintha a *ScanDisk* lemezellenőrző futna más szöveggel, lásd *3. ábra*), sőt mivel az egész nagyon általánosan lett megírva, mi is írhatunk egy sajátot (az alaptémákat célszerű tanulmányozni), amelyet hasznos függvények segítenek (színkezelés, kurzor pozicionálása, stb.). A témákat ki is lehet próbálni (még bootolás előtt) a

```
TestBootTheme Theme_Neve
```

paranccsal. A *3. ábra* is így készült.

Összefoglalás

A disztribúció teljes bemutatása túllépne a cikk terjedelmén, és nem is ez volt a cél, hanem csak figyelemfelkeltés és kedvcsinálás.
A terjesztésnek sok hasznos és érdekes tulajdonsága, beállítási lehetőségei vannak, igazi „bütykölős" rendszer, olyannyira, hogy például még a véletlenszám-generátor inicializálását is nekünk kell megírni (a részletekkel kapcsolatban `man urandom`), tehát kezdőnek egyáltalán nem ajánlható (a *FAQ*-ban szólnak is erről).
Eddig elsősorban az előnyökről volt szó, azért essen szó a hátrányairól:

- A `SymlinkProgram` és a `DisableProgram` a *libexec* könyvtárat nem kezeli, és egy-két sorral mindkettőt bővíteni kell, ahelyett, hogy lenne egy konfigurációs fájl, amiben tárolná, hogy melyik könyvtár tartalmát hova linkelje (amit a saját *Linux From Scratch* rendszeremben meg is valósítottam).
- A csomagkezelés részleteiből kifolyólag nincsenek „programtípusok", tehát nem tudjuk eldönteni egy programról, hogy ő játék vagy felhasználói program vagy valami más (ez is meg van valósítva az *LFS*-emben).
- Sok program nem szereti, ha az állományok nem ott vannak, ahol ő feltételezi vagy pedig a szimbolikus láncokat nem kezeli (ilyen a `makewhatis` program, amit egy kis átírás után rábírhatunk a szimlinkek felvételére is).
- Valahol nagyon általánosan (absztraktul) vannak megírva a szkriptek (például bootfolyamat), valahol pedig gyakorlatilag „beégetve" minden (csomagkezelő szkriptek). Véleményem szerint az előbbi alternatíva a jobb, de a vegyes megoldás a lehető legrosszabb, mivel így a rendszer nem tűnik egységes egésznek.
- A források keresése/letöltögetése nehézkes (mindent nekünk kell megkeresni), függőség-kezelés gyakorlatilag nincs (ellentétben például a *Gentoo* ebuildjeivel), ami internet nélkül egy kicsit keserves. Viszont ez a „Csináld magad" mozgalom híveinek inkább előny.

Összességében egy nagyon jól használható, nagyon érdekes és ami sokak számára fontos, nagyon gyors (ami leginkább a forrás-alapnak köszönhető) disztribúciót ismerhetünk meg a *GoboLinux* személyében.

Udvari Zsolt
(udvzsolt@gmail.com)
25 éves vagyok, egy gimnáziumban tanítok matematikát és fizikát. A Linuxszal először 2004 elején találkoztam, az UHU Kamionja volt, ami elgázolt. Azóta 4-5 disztribúciót hosszabban is használtam, jelenleg egy saját építésű LFS-t nyúzok.