Változatkezelés az Arch használatával: bevezetés

Ha a CVS-t akarjuk felváltani, vagy komolyan gondolkozunk egy változatkezelő rendszer bevezetésén, íme egy hatékony eszköz, amely nyomon követi a változtatásokat és kézben tartja a projektjeinket.

Az ***Arch*** rohamosan válik az egyik leghatékonyabb eszközzé a szabad szoftvereket fejlesztő programozók eszköztárában. Ez a cikk annak a három részes cikksorozatnak az első tagja, amely bemutatja az ***Arch*** használatát az osztott fejlesztés során, valamint megismerteti az olvasót a megosztott archívumok és az ***Arch projektekhez*** alkalmazható önműködő parancsfájlok használatával.
Ebből a cikkből megtudhatjuk, hogyan szerezhetjük meg a kódot egy nyilvános Arch-archívumból, hogyan tölthetjük vissza a változásokat és hogyan készíthetünk helyi fejlesztési ágat a projektről kapcsolat nélküli használatra. Megismerhetünk továbbá néhány technikát a helyi és távoli archívumokkal való hatékonyabb munkára is.

A változatkezelés története

A változatkezelés egy projekt változásainak nyilvántartásával foglalkozik. A szabad szoftverek fejlesztése során alapvető fontosságú a projekten végzett munka elemzésének, a fejlesztési ágak összehasonlításának, a változtatások ismétlésének vagy visszavonásának lehetősége. A lehetséges közreműködők nagy száma és a változtatások gyors kibocsátása rövid időn belül életre hívta ezeknek a változtatásoknak a kezelésére szolgáló eszközöket.
A változáskezelés legősibb eszköze a szalagos tároló volt. Egy projekt korábbi változatait a szalagon tárolt biztonsági mentésekből kellett előhúzni és soronként összehasonlítani az új változattal. A szalagról történő beolvasása nem egy gyors folyamat, így ez semmilyen szempontból nem nevezhető hatékony megoldásnak.
Ennek a hátráltató tényezőnek a kiküszöbölésére sok fejlesztő kéznél tartotta a fájlok régebbi változatait is összehasonlítás céljából, és ez a lehetőség hamarosan a fejlesztőeszközökben is megjelent. A fájlalapú változatkezelés, amelyet például az ***Emacs*** karakteres szerkesztő is használ, számozott biztonsági másolatokat őriz a fájlokról, így ha meg akarjuk vizsgálni a változtatásokat, könnyedén összehasonlíthatjuk a *foo.c~7~* fájlt a *foo.c~8~* fájllal. Néhány korai kereskedelmi operációs rendszerben még a fájlrendszer szintjén is megjelentek ezek a számozott biztonsági másolatfájlok.
Közel két évtizeden keresztül a szabad programok külső fejlesztői által előszeretettel alkalmazott formátum a foltfájl volt, amit különbségfájlként is emlegettek. Két fájlt összehasonlítva egy erre a célra fejlesztett különbségképző program állította elő azt a listát, amely a két fájl eltéréseit emelte ki. A kimenetként előálló különbségfájl által tartalmazott eltérések jóváhagyásához csak egy foltozó programon kellett lefuttatni a fájlt.
A 90-es évektől a ***CVS (Concurrent Versions System)***, párhuzamos változatkezelő rendszer) vált a fejlesztők egy központi csoportja által alkalmazott alapértelmezett változáskezelő rendszerré. Egy egyszerű ágkövető és egyesítő rendszer teszi lehetővé a felhasználó számára a különböző fejlesztési ágakkal történő kísérletezést, majd a sikeres próbálkozások főprojektben történő rögzítését.
A ***CVS***-nek megvannak a maga korlátai, amelyek sok projekt számára egyre inkább terhet jelentenek. Először is a rendszer egyáltalán nem tárolja az olyan metaadatok megváltozását, mint a fájlokhoz kötődő jogosultságok vagy a fájl nevének a megváltozása. Ilyen hiányosság az is, hogy a bejegyzések nem kapcsolhatók össze, így igen nehézkes a több fájlon vagy könyvtáron átívelő változások követése. Végül majdnem minden távoli ***CVS***-táron végrehajtott művelet egy újabb kapcsolat megnyitását igényli a kiszolgáló felé, megnehezítve ezzel a kapcsolat nélküli munkát.
A ***Subversion Projekthez*** hasonló erőfeszítések nagy utat tettek meg a ***CVS***-ben fellelhető hiányok kiküszöbölése felé vezető úton. A ***Subversion*** lényegét tekintve egy ***CVS++***, amely támogatja a metaadatok változásának rögzítését és az elemi bejegyzéseket. Továbbra is szükség van azonban egy olyan központi kiszolgálóra a hálózaton, amelyhez az összes változáskezelésben részt vevő ügyfélgép csatlakozik.

A megosztott változáskezelő rendszerek

Az utóbbi néhány évben a változáskezelő rendszereknek egy új generációja látott napvilágot, amelyek mindegyike az osztott modellt alkalmazza. Az osztott változáskezelő rendszerek szakítanak a központosított tárhely elvével, ehelyett az egyenrangú gépek alkotta elrendezést részesítik előnyben. Minden fejlesztő fenntartja a maga tárolóhelyét, a rendelkezésre álló eszközök pedig lehetővé teszik a végrehajtott változtatások egyszerű átadását a hálózaton lévő gépek között.
A ***Monotone***, a ***DARCS*** az ***Arch*** és a hasonló projektek olyan környezetben tettek szert nagy népszerűségre, ahol a szabad forráskódú fejlesztés a jól összekapcsolt egyetemeken kívül zajlik, és a noteszgépek sokkal elterjedtebbek.

Jelen pillanatban az osztott rendszerek egyik legígéretesebb darabja a ***GNU Arch***. Az ***Arch*** úgy bonyolítja a kapcsolat nélküli használatot, hogy a felhasználókat helyi másolatok létrehozására ösztönzi és hatékony eszközöket kínál a projektek archívumok közti mozgatására. Az ***Arch***-ból hiányzik bármiféle kizárólagos kiszolgálói folyamat, az archívum kezelésére pedig a fájlrendszer-műveletek egy hordozható részhalmazát használja. Az archívumok egyszerűen olyan könyvtárak, amelyeket a hálózaton az általunk választott fájlrendszer-protokoll segítségével teszünk elérhetővé. Mi több, az ***Arch*** támogatja az archívumok ***HTTP***, ***FTP*** vagy ***SFTP*** protokollokon keresztül történő elérését. Az egyik nagy előnye annak, hogy nincs egy kitüntetett démon, hogy nem kap egy új program jogokat a kiszolgáló gépünkön. Így a biztonsággal kapcsolatos kérdések csak az ***SSH*** démonnal vagy a webkiszolgálóval lesznek kapcsolatban, amin a legtöbb rendszergazda amúgy is rajta tartja a szemét. Egy másik előny, hogy az ***Arch*** használatakor a legtöbb feladat megoldásához nincs szükség a root jogosultságaira. A fejlesztők elkezdhetik csak a saját gépeiken használni és archívumokat tehetnek közzé anélkül, hogy akár csak telepíteni kellene az ***Arch***-ot a webkiszolgáló gépen. Ez hatással van az alkalmazás módjára is: míg a ***CVS*** vagy a ***Subversion*** használata egy felülről lefelé irányuló döntés, amely a teljes projektcsapatra vonatkozik, az ***Arch***-ot alkalmazhatja csupán egy-két fejlesztő is a csapatban, míg át nem áll a többi tag is.

Hogyan juthatunk a tla-hoz?

Az ***Arch*** eredetileg ***Tom Lord hackerlab*** programkönyvtáraihoz tartozó burkoló- és héjprogram-gyűjtemény volt. Abban az időben a programot larch néven emlegették és egy kicsit esetlenül lehetett használni. Az ügyfél ***C*** nyelven teljesen át lett írva, a neve pedig ***tla*** lett ***(Tom Lord's Arch)***. A felülete még mindig nem tökéletes, de arra megfelel, hogy egy képzett fejlesztő normál módon használja a napi munkájához. A ***tla*** csomagjai a legtöbb ***GNU/Linux*** rendszercsomaghoz elérhetőek (források: ➲ ***www.linuxjournal.com/article/7752***).

Egy csak olvasható projekt letöltése

Miután feltelepítettük a ***tla***-t, nem árt, ha valamilyen letöltött kóddal ki is próbáljuk. Az ***Arch*** az adatainkat egy könyvtárban tárolja, amit archívumnak nevezünk. Egy archívumon belül az adataink egymásba ágyazott kategóriákban helyezkednek el: ezek a projektek (ez az egész munkának a neve), az ágak (a fejlesztés vagy valamilyen egyéb leíró elem egy bizonyos ága, és a változatok (egy egyszerű szám, amelyet annak a jelölésére használhatunk, hogy egy bizonyos szál fejlesztésében milyen messze sikerült előrehaladnunk). A kódhoz jutás első lépése egy nyilvános archívum regisztrálása, így az ***Arch*** egy nevet tud rendelni az archívum helyéhez:

```
$ tla register-archive http://www.lnx-bbc.org/arch
```

Ezt követően a `tla archives` parancs futtatása után látnunk kell az `lnx-bbc-devel@zork.net--gar` archívumot Ha kíváncsiak vagyunk, hogy itt milyen projektek tárolása történik, a teljes lista lekérésére használhatjuk a ***tla abrowse*** parancsot.

```
$ tla abrowse lnx-bbc-devel@zork.net--gar
lnx-bbc-devel@zork.net--gar
```

```
lnx-bbc
  lnx-bbc--research
    lnx-bbc--research--0.0
      base-0 .. patch-10

  lnx-bbc--stable
    lnx-bbc--stable--2.1
      base-0 .. patch-29

scripts
  scripts--gargoyle-bin
    scripts--gargoyle-bin--1.0
      base-0 .. patch-7
```

Ebből a listából az látható, hogy a `lnx-bbc-devel@zork.net--gar` archívum két projektet tartalmaz `lnx-bbc` és `scripts` néven. Az `lnx-bbc` két ággal rendelkezik: `research` (kísérleti) és `stable` (stabil). Az `lnx-bbc--research` ágnak csak egy változata van (0.0), amelynek tíz változtatását tartalmazza az archívum. Az `lnx-bbc--stable` egy változattal (2.1) és 29 változással szerepel.
Mivel most már rendelkezünk az `LNX-BBC` regisztrált archívumával a helyi listánkban, nincs akadálya annak, hogy letöltsük az `LNX-BBC` stabil ágának egy másolatát:

```
$ tla get \
lnx-bbc-devel@zork.net--gar/lnx-bbc--stable lnxbbc
```

Amikor befejeződött a letöltés és a foltok alkalmazása, kell lennie egy ***lnxbbc/*** könyvtárunknak tele mindenféle fájllal. Tegyünk úgy, mintha megváltoztatnánk a kódot: lépjünk be az ***lnxbbc/*** könyvtárba és írjunk be valahova egy új megjegyzést.

Változtatások hozzáadása

Miután megváltoztattuk a fájlt, a `tla what-changed` parancsnak ki kell írnia egy `M robots.txt` sort, jelezve, hogy a `robots.txt` tartalma megváltozott. A változás részleteit a `tla what-changed --diffs` paranccsal kérhetjük le, amely kilistáz egy különbségfájlt készen arra, hogy visszaküldjük a projekt fejlesztőcsapatának.

```
--- orig/robots.txt
+++ mod/robots.txt
@@ -1,3 +1,5 @@
+# Welcome, robots!
+
 User-agent: *
 Disallow: /garchive/
 Disallow: /cgi-bin/
```

Ennek a módszernek a hátránya, hogy a különbségfájl nem jelzi a metaadatokban bekövetkezett változásokat: az eltávolított és új fájlok nem fognak megjelenni, amikor egy másik fejlesztő lefuttatja a különbség-foltunkat. Ahhoz, hogy a projekt kezelőinek egy bonyolultabb változtatást át tudjunk adni, egy változáskészletet (changeset) kell létrehoznunk. Az ***Arch***-ban egy változáskészlet a teljes könyvtárszerkezetet jelenti a fájlok változásainak, a foltoknak, az új és eltávolított fájloknak a teljes nyilvántartásával. A közreműködés

legjobb módszere az, ha létrehozunk egy változáskészlet-könyvtárat és ezt összecsomagoljuk a továbbításhoz:

```
$ tla changes -o ,,new-robot-comment
$ tar czvf my-changes.tar.gz ,,new-robot-comment/
```

Az *Arch* nem veszi figyelembe a két veszővel, egyenlőségjellel és még néhány különleges karakterrel kezdődő fájlneveket. Azzal, hogy elhelyezzük a „ jelet a változáskészletünk könyvtárnevének az elején, elkerüljük, hogy az *Arch* kifogásolja, hogy az új könyvtárunk még nem létezik az archívumban. A gyakorlatban hasznos lehet az elektronikus levélcímünk vagy valamilyen más azonosítónk feltüntetése a tar-csomag fájlnevében és a változáskészlet könyvtárának a nevében.

Az archívum frissen tartása

Időnként szükségünk lesz a projekt legfrissebb változtatásainak letöltésére. Ez nem jelent bonyolultabb feladatot, mint a `tla update` parancs futtatását a letöltött másolatban. Az *Arch* először egy `tla undo` parancs végrehajtásával biztonságba helyezi a helyi változtatásainkat, mielőtt az új változáskészleteket alkalmazná. Miután feltette az összes foltot, lefuttatja a `tla redo` parancsot a helyi változások visszaállítására.
A korábban bemutatott minden `tla` parancshoz szükség van egy élő hálózati kapcsolatra az archívumot tároló `lnx-bbc.org` rendszerhez. A kapcsolat nélküli használathoz létre kell hoznunk egy helyi archívumot és ezen belül egy fejlesztési ágat.

Egy archívum létrehozása

Mielőtt dolgozni kezdenénk egy írható-olvasható archívumban, azonosítanunk kell magunkat a `tla` számára:

```
$ tla my-id "J. Random Hacker <jrh@zork.net>"
```

A levélcímünk megadása után elérkezett az idő, hogy archívumot készítsünk a projektjeink számára. Az *Arch* több archívum létrehozását is lehetővé teszi, de egy archívumon belül is tetszőleges számú projekt és fejlesztési ág tárolására van lehetőség.
Az archívumok neve két részből áll, amelyet kettős kötőjellel választunk el egymástól: az első rész az elektronikus levélcímünk, a második pedig valamilyen azonosító. Sok fejlesztő azonosítóként az évszámot adja meg és minden évben új archívumot kezd:

```
$ tla make-archive -l jrh@zork.net--2004 ~/ARCHIVE
$ tla my-default-archive jrh@zork.net--2004
```

A `my-default-archive` paranccsal megadva az alapértelmezett archívumunkat a helyi archívumon végrehajtandó műveletek begépelését könnyíthetjük meg.

Fejlesztési ág létrehozása

Az *Arch* arra ösztönzi a felhasználókat, hogy a fejlesztési ágak használatával ágaztassák el és fésüljék össze a projekteket. Az ágak jelentik az elsődleges módszert a kód egyik archívumból a másikba történő mozgatására akár hálózaton keresztül is.
A fejlesztési ágakat használhatjuk a projekt egy teljesen új fejlesztési irányát tartalmazó teljes kód változásának nyomon követésére, de alkalmazhatjuk arra is, hogy a projektünk egy átmeneti másolatát hozzuk létre a noteszgépünkön, amelyen hálózati kapcsolat nélküli környezetben is dolgozhatunk.
A nyilvánossá tett ágak jelentik a fejlesztéssel kapcsolatos párbeszéd elsődleges eszközét is az *Arch*-ot használó fejlesztők között. Ahelyett, hogy levélben küldözgetnének nagy méretű tar-csomagokat egymásnak, a közreműködő valószínűleg inkább egy ágat hoz létre a helyi változások tárolására, és arra kéri fel a fejlesztőket, hogy fésüljék bele ezeket a változtatásokat a fő projektbe. Itt ragyog fel leginkább az *Arch* decentralizált és demokratikus természete: bárki csatlakozhat a fejlesztéshez anélkül, hogy ehhez a központi fejlesztőcsapattól valamilyen különleges jogosultságot kellene kapnia.
Mielőtt elágaztatnánk az `lnx-bbc` projektet, helyet kell biztosítanunk a projekt számára az archívumunkban. A projekt azonosítója az archívum nevéhez hasonlóan épül fel: a kategória (vagy projektnév), két kötőjel, az ág neve, két kötőjel és a verziószám. Valószínűleg *Tom Lord LISP*-szakértői tapasztalata vezetett ezeknek a kötőjeleknek a használatához:

```
$ tla archive-setup lnx-bbc--robot-branch--0.0
```

Ezzel létrejön az `lnc-bbc` nevű kategória a `robot-branch` nevű fejlesztési ággal `0.0` verziószámmal. A `jrh@zork.net--2004/` archívum-nevet nem kell megadnunk a projekt neve előtt, mivel ez az alapértelmezett archívumunk.

A fejlesztési ágak címkézése

Végül elérkezett az idő arra, hogy felcímkézzük az águnkat. Ez annyit jelent, hogy a `robot-branch` egy címkével kezdődik, amely az `lnx-bbc-devel@zork.net--gar` archívumban lévő egy projekt adott változatára mutat, a helyi változtatások pedig ettől a ponttól kezdődnek:

```
$ tla tag \
  lnx-bbc-devel@zork.net--gar/lnx-bbc--stable--2.1 \
  lnx-bbc--robot-branch--0.0
```

Ha itt futtatjuk a `tla abrowse` parancsot, az archívumunkról a következő képet kell mutatnia:

```
jrh@zork.net--2004
  lnx-bbc
    lnx-bbc--robot-branch
      lnx-bbc--robot-branch--0.0
        base-0
```

Az új fejlesztési ág használatba vétele

Most már készen állunk arra, hogy letöltsük az új águnk egy másolatát:

```
$ tla get lnx-bbc--robot-branch robot-branch
```

Itt beléphetünk a `robot-branch` könyvtárba és módosíthatunk egy-két dolgot:

```
$ chmod 444 index.txt
$ tla mv faq.txt robofaq.txt
$ echo "ROBOT TIME" > robot-time
```

```
$ tla add robot-time
$ tla rm ports.txt
```

A `tla mv` parancs olyan módon módosítja egy fájl nevét, hogy az ***Arch*** képes legyen a fájl változásának nyomon követésére. Fontos, hogy ezt a parancsot használjuk a szabványos `mv` helyett. A `tla add` parancs előkészíti egy fájl hozzáadását az archívumhoz, a `tla rm` pedig egy fájl eltávolításának tervét rögzíti. Ezek a változások most már megjeleníthetők a helyi fejlesztési ágban:

```
$ tla commit
```

Elindul a kedvenc karakteres szerkesztőnk (a korábban a `$EDITOR` környezeti változóban megadottaknak megfelelően) egy végrehajtási jegyzőkönyv sablonját kínálva. Miután kitöltöttük a bejegyzéseket, a mentéssel és kilépéssel véglegesítjük a változtatásokat.
A `tla abrowse` parancsot kiadva láthatóvá válik, hogy most már két változata létezik az archívumunkban lévő `robot` ágnak, a ***base-0*** és a `patch-1`:

```
jrh@zork.net--2004
  lnx-bbc
    lnx-bbc--robot-branch
      lnx-bbc--robot-branch--0.0
        base-0 .. patch-1
```

Egy projekt összefésülése két különböző archívumból

Természetesen, mialatt a saját águnkon dolgozunk, az eredeti archívumban is folytatódhat a munka. A `tla update` parancs futtatása csak a helyi ágból olvassa ki a változtatásokat, az eredeti projektből nem. A központi archívumból a `star.merge` paranccsal tudjuk a változtatásokat átemelni:

```
$ tla star-merge \
lnx-bbc-devel@zork.net--gar/lnx-bbc--stable--2.1
```

Ütközések esetén (amikor a saját águnk és a központi projekt is tartalmaz változtatásokat ugyanazokra a kódsorokra vonatkozóan) az ***Arch*** az eredeti foltozási eljárást használja, amelynek során létrehozza a megfelelő `.orig` és `.rej` kiterjesztésű fájlokat. Érdemes egy keresést végrehajtani a visszautasításokat tartalmazó fájlokra, mielőtt a star-merge parancs változtatásait véglegesítenénk.

Az archívum-műveletek felgyorsítása

Észrevehettük, hogy az egyes változatok vagy `base-0` vagy `patch-#` néven szerepelnek, ahol a # jelenti a `base-0`-ra alkalmazandó változtatások sorszámát. Az ***Arch*** napló-rendszerű archiválási formátumot alkalmaz, így rögzíti a projekthez kötődő bármilyen információ változását. Ez azt is jelenti, hogy a nagyméretű, sok változattal rendelkező projektek esetén bizonyos feladatok végrehajtása hosszú időt vehet igénybe. A műveletek meggyorsítására pillanatképet készíthetünk egy adott változatról. Az ***Arch*** pillanatképei egy közzétett változatról készült egyszerű tömörített tar-csomagok. Amikor valamilyen műveletet végrehajtunk, az ***Arch*** megkeresi a legmagasabb sorszámmal rendelkező pillanatképet és a szükséges változtatásokat ettől kezdve hajtja végre:

```
$ tla cacherev
```

Ennek befejezése után futtathatjuk a `tla chachedrevs` parancsot, hogy lássuk, milyen pillanatképekkel rendelkezünk az archívumunkban:

```
lnx-bbc--robot-branch--0.0--base-0
lnx-bbc--robot-branch--0.0--patch-1
```

A könyvtárak

Mivel nem mindig rendelkezünk megfelelő jogosultsággal az archívumban a pillanatképek készítéséhez, hasznos lehet helyi képeket létrehozni a fájlműveletek felgyorsítása érdekében. Az ***Arch*** egy másik fajta pillanatképet is támogat, amelyet könyvtárnak (library) nevez, s amely a különböző ágakból származó közzétett fájlokat tartalmazza. Ez különösen a távoli archívumoknál hatékony, mivel így még egy változat alapképét sem kell letöltenünk ahhoz, hogy alkalmazzuk a változáscsomagunkat:

```
$ mkdir ~/LIBRARY
$ tla my-revision-library ~/LIBRARY
$ tla library-config --greedy ~/LIBRARY
$ tla library-add \
  lnx-bbc-devel@zork.net--gar/lnx-bbc--stable--2.1
```

Ez a könyvtár nem éppen kis méretű, a fenti példához tartozó csomag 78 MB méretű volt 2004 júniusában, de egy lassú kapcsolat esetén a dolog bőven megéri a fáradságot. Ráadásul a hordozható gépek gyakran lassú ATA merevlemezes egységgel rendelkeznek, az archívummal kapcsolatos műveletek pedig nagy terhelést jelenthetnek, mivel az IDE meghajtók sok processzoridőt igényelnek.
Egy önműködően frissülő Arch-könyvtár gyorsabbá és reaktívabbá teszi változáskezelő műveleteinket még a helyi archívumok esetén is.

Folytatás következik

A sorozat következő cikkében azt fogjuk megtanulni, hogyan készíthetünk nyilvánosan elérhető tükrözéseket, amellyel a központi fejlesztők a fejlesztési ágainkból visszaemelhetik a változásokat. Azt is megtanuljuk, hogyan válogathatunk egy sűrűn változó ág változáskészleteiből, és hogyan tudjuk a változáskészleteinket biztonsági célból kriptográfiai módszerrel megjelölni a ***GnuPG*** segítségével.
A sorozat harmadik, befejező részében az ***Arch*** központi fejlesztéseket támogató eljárásairól lesz szó. Megtanuljuk, hogyan kezelhetünk megosztott hozzáférésű archívumokat az ***OpenSSH SFTP*** protokolljával, és hogyan írhatunk parancsfájlokat az archívum feladatainak önműködő végrehajtására.

Linux Journal 2004. november, 127. szám

Nick Moffit, Linux szakértő, Sun Francisco Baynegyedében él. A GNU/Linux LNX-BBC Bootable Business Card csomagjának szerkesztő mérnöke, valamint a GAR szerkesztőrendszer szerzője. Amikor nem a rendszereivel foglalkozik, a városi tömegközlekedés történetét tanulmányozza.