Verziókezelés az Arch-al (2. rész)
Karbantartás és kifinomult használat

Helyezzünk üzembe hatékony verziókezelő rendszert kiszolgálónkon, mindössze a web és az SSH programok használatával. Bemutatjuk, mi szükséges egy program-projekt Arch alapú karbantartásához.

Az ***Arch***, az egyik mostanában megjelenő fiatal verziókezelő, fontos szerkezeti előnyt képes felmutatni az jó öreg ***Concurrent Version System (CVS)*** rendszerrel és társaival szemben. Decentralizált verziókezelő lévén, az ***Arch*** segítségével nagy mennyiségű fejlesztési munkát végezhetünk különleges előjogok igénylése nélkül. Az ***Arch*** ezen felül hatékony archívumközi műveleteket is nyújt, melyek jelentkezésre ösztönzik a harmadik felet.

A sorozat előző részében bemutattuk az alap ***Arch*** műveleteket: hogyan kérjünk ki kódot valamint miképpen lehet ágakat készíteni a távoli archívumokból. Most megtudhatjuk, hogyan vonhatjuk vissza a módosításainkat, hogyan adhatjuk ki saját archívumainkat nyilvános tükrökön keresztül és hogyan másoljuk változtatásunkat archívumból archívumba ha elfelejtettünk új ágat csinálni.

Az ***Arch*** program neve `tla`. Az `arch` programnév ugyanis már foglalt a ***POSIX*** szabványban, amely kiköti, hogy a ***/bin/arch*** parancsnak rendszerinformációkat kell visszaadnia. A `tla` súgójának futtatásával sok információhoz juthatunk. Amennyiben valamelyik parancs, például a `commit` paramétereire vagyunk kíváncsiak, érdemes lefuttatni a

```
tla commit -H
```

parancsot, hogy lássuk mit is tud a `tla commit` parancsa.

Változások mentése

Minden verziókezelő rendszer egyik nyilvánvaló előnye, hogy visszavonhatjuk egy vagy több változtatásunkat. Mindenki követ el hibákat, nem is ritkán, így aztán elég fontos hogy eszközeink rendelkezzenek valamilyen barátságos helyreállítási lehetőséggel.

A kikért fa helyi változtatásainkat nem tartalmazó változatához legkönnyebben a `tla undo` futtatásával térhetünk vissza. A parancs létrehoz egy ***undo-1/*** nevű könyvtárat, ahol az összes módosításunk található. Ha úgy tetszik, a `tla redo` parancs kiadásával egyszerűen kiadjuk újra érvényesíthetjük változtatásainkat. Például:

```
$ tla register-archive http://www.lnx-bbc.org/arch
$ tla get \lnx-bbc-devel@zork.net--gar/
  ➥lnx-bbc--stable bbc
$ cd bbc/
$ echo "BIG MISTAKE" > robots.txt
$ echo "#smaller change" >> Makefile
$ tla undo
$ tla redo
```

A `tla undo` parancs legnagyobb hasznát a „na-álljunk-csak-meg-egy-pillanatra" esetekben vehetjük, amikor a jelenleg folyó munkát kicsit félretennénk, valamilyen gyors változtatás érdekében. Az ***Arch*** belsőleg is használja az `undo` és `redo` parancsokat például az `update` vagy `star-merge` utasítások végrehajtásakor.

Egyetlen fájl visszaállítása

Amennyiben a hiba egyetlen állományra korlátozódik, nincs szükség a teljes változáshalmaz elmentésére. Az ***Arch*** egyetlen fájl visszaállításának problémáját úgy oldja meg, hogy az utolsó frissítés óta bekövetkezett változásokról egységes ***diff*** állományt készít. A ***diff*** aztán betölthető a visszaállítás módba kapcsolt ***patch*** programba, így a változtatások eltávolíthatók az állományból.

```
$ tla file-diffs robots.txt | patch -R
```

Amennyiben az állományt véletlenül letöröltük, a parancs végrehajtás előtt meg kell hívnunk a `touch robots.txt` parancsot. Állomány nélkül (még csak egy üres állományról van is szó), az ***Arch***-nak nincs mire támaszkodnia a ***file-diffs*** létrehozásánál. Ugyanakkor ha teljes változáskészletekkel dolgozunk az ***Arch*** sokkal intelligensebb.

Egész változtatáskészletek visszaállítása

Az ***Arch*** egyik nagy előnye elődjéhez a ***CVS***-hez képest, hogy támogatja a változtatás-készletek létrehozását és kezelését. A változtatás-készlet tulajdonképpen egyetlen

`tla commit` hívás során lejegyzett valamennyi szerkesztés, átnevezés, felvett és törölt állományok és naplóbejegyzések adatainak összessége.
Előfordul, hogy olyan változáskészletet küldünk el, amit nem kellett volna, esetleg valaminek csak az ideiglenes állapotát mentjük el, míg egy stabilabb változat betölti a helyét. Ezekben az esetekben visszavonhatjuk a változtatásokat ha fordított sorrendben játsszuk le őket:

```
$ tla replay --reverse \
  jrh@zork.net--projects/foo--bar--1.0--patch-4
$ tla sync-tree \
  jrh@zork.net--projects/foo--bar--1.0--patch-4
```

Az első parancs visszavonja az foo fa bar ágában található 1.0-ás verzió negyedik változtatáskészletét, még akkor is ha az nem a legfrissebb verzió. Mivel az adott változtatáskészlethez a naplóbejegyzéseket is mentjük, a `tla sync-tree command` parancs a `commit` naplót elvárásainknak megfelelően állítja vissza.
A ***patch-4*** változáskészlet továbbra is megtalálható a ***jrh@zork.net--projects*** archívumban, a fát tehát továbbra is ki lehet kérni e szerint az állapot szerint. A fenti parancsok kizárólag a jelenlegi munkaváltozatra voltak hatással. Amikor a fenti felhasználó lefuttatja a `tla commit` parancsot, új változáskészlet kerül fel, amely tartalmazza a ***patch-4***-et is.

Szemezgetés más ágakból

A `tla replay` parancs egyszerű „mégsem” funkciónál hasznosabb műveltekre is felhasználható. Az ***Arch*** egyik legvonzóbb tulajdonsága, hogy segítségével távoli források változáskészletei között szemezgethetünk anélkül, hogy a nekünk nem tetsző változásokat telepítenünk kellene. Vegyünk egy projektet, amelyet ***Bob*** tart karban. A projekt stabil ága (***foo--stable***) és kísérleti ága (***foo--experimental***) is ***Bob***-nál található. Az összes kiadást a legfrissebb ***branch--foo--stable--2.4.2*** stabil ágból készítjük. A kísérleti ágban tesszük közzé valamelyest hivatalosabb formában a kalandos próbálkozásokat.
Alice valamelyik kísérleti kódon szeretne dolgozni, ezért a munkához saját helyén kijelöli ***Bob*** kísérleti ágát:

```
$ tla my-id “Alice B. Hacker <abh@zork.net>”
$ tla make-archive -l abh@zork.net--work \
  sftp://abh@zork.net/home/abh/public_html/arch
$ tla archive-setup foo--hackery--0.0
$ tla register-archive
http://entar.net/~bob/fooarch
$ tla tag \
  bob@entar.net--code/foo--experimental--0.0 \
  abh@zork.net--work/foo--hackery--1.0
```

Miközben a kísérleti verzión dolgozik, ***Alice*** felfedez egy hibát ami fölött ***Bob*** tekintete úgy tűnik átsiklott. A javítás egyszerű, ezért félreteszi a jelenlegi munkáját a tla undo paranccsal és feltölti a javítást:

```
$ tla undo
$ vi buggy_file.c another_buggy_file.c
$ tla commit
M  buggy_file.c
M  another_buggy_file.c
* committed
  abh@zork.net--work/foo--hackery--1.0--patch-9
$ tla redo
```

Alice hamarosan befejezi a változtatásokat és jelzi ***Bobnak*** hol találhatóak az archívumai. ***Bob*** úgy dönt a változtatások elfogadhatóak a kísérleti kódágba és a `star-merge` funkcióval beolvasztja:

```
$ tla get bob@entar.net--code/
➥foo--experimental--0.0
$ cd foo--experimental--0.0/
$ tla register-archive http://zork.net/~abh/arch/
$ tla star-merge \
  abh@zork.net--work/foo--hackery--1.0
```

Miközben ***Alice*** változásnaplóját olvasgatja, ***Bob*** ráébred, hogy a kijavított hiba a stabil verzióban is létezik. Minthogy nem szeretné a teljes kísérleti kódot kiemelni a ***hackery*** ágból, ***Bob*** szemezgetve csak a hibajavítást végző változáskészletet tölti át:

```
$ tla get bob@entar.net--code/foo--stable--2.4.2
$ cd foo--stable--2.4.2/
$ tla replay \
  abh@zork.net--work/foo--hackery--1.0--patch-9
```

Változáskészletünk terjesztése

Alice és ***Bob*** együtt tudtak dolgozni, annak ellenére, hogy a két fejlesztő nem osztozott semmilyen, mindkettőjük számára elérhető közös rendszeren. Egyik fejlesztő sem készített dedikált kiszolgálót; egyszerűen a szokásos elterjedt protokollokat használták a ***http***-t, ***SSH***-t és az ***SFTP***-t. ***Alice*** archívumának előnye, hogy web könyvtárként elérhető az Interneten keresztül, akárcsak ***Bob*** hivatalos archívuma.
Az ***Arch*** segítségével ***Alice*** és ***Bob*** képesek voltak egymás egyedi archívumán és a különbségeken dolgozni, miközben semmi különlegeset nem használtak csak az ***Apache*** és az ***OpenSSH*** kiszolgálókat.

Aláírások

Ilyen sok kód küldözgetése az interneten keresztül mindig is zavarta kicsit a szabad program fejlesztőket, még ha csak tudat alatt is. A jelenlegi szakértői felülvizsgálati (peer review) rendszer gyorsan és hatékonyan megoldotta a rossz szándékú kódküldés problémáját, de még jobb lenne ha minden változáskészlet küldőjét kétséget kizáróan azonosítani lehetne.
A fejlesztők az ***Arch*** segítségével titkosítva aláírhatják a változtatáskészleteiket, így a saját megbízhatósági hálózatukon keresztül meg lehet állapítani a küldő kilétét. Habár ez nem feltétlenül bizonyíték a fejlesztő jó szándékára nézve, a hamisított küldeményeknek legalább gátat szab.
Ha az ***Arch***-ban titkosítást szeretnénk használni, először is készítenünk kell egy ***GnuPG*** kulcsot.

```
$ gpg --gen-key
```

Sajnos az aláírt archívumok némiképpen eltérő funkciókat kívánnak mint az aláírás nélküli változatok. Ezért külön archívumot kell fenntartanunk az aláírt küldeményeknek. A *tla make-archive* parancsot a `-s` kapcsolóval futtatva *GnuPG* aláírások tárolására is alkalmas archívumot kapunk:

```
$ tla make-archive -ls jrh@zork.net--signed \
  ~/SIGNED-ARCHIVE
$ tla my-default-archive jrh@zork.net--signed
```

Végül, néhány beállítás állományt kell létrehoznunk, hogy az *Arch* alá tudja írni a változtatás készleteket és ellenőrizhesse az aláírásokat. Először is a *tla* terjesztésében található *gpg-check.awk* nevű *awk* parancsfájlt kell feltelepítenünk valahová az *Arch*-ot futtató rendszeren. A *Debian tla* csomagok a */usr/bin/tla-gpg-check* helyre telepítik alapértelmezés szerint. Ha szeretnénk, hogy az *Arch* ellenőrizni is tudja az aláírásokat, a `~/.arch-params/signing/=default.check` állományba egyetlen sort kell felvinnünk:

```
$ mkdir ~/.arch-params/signing/
$ echo \
 ´tla-gpg-check gpg_command="gpg --verify-files
 ➥-"´\
  > ~/.arch-params/signing/\ =default.check
```

Amennyiben azt szeretnénk, hogy a kulcsok automatikusan töltődjenek le egy nyilvános kulcskiszolgálóról, akkor kiegészíthetjük a `gpg_command` parancsot például a `--keyserver pgp.mit.edu --keyserver-options auto-key-retrieve` kapcsolókkal. Mostantól az *Arch* minden `get` vagy `update` műveletek során szükség szerint letölti a *pgp.mit.edu* címről az archívum aláírásokat majd ellenőrzi őket.
Amennyiben szeretnénk ha az *Arch* automatikusan aláírná a változtatáskészleteket amelyeket -s kapcsolóval készített archívumokba küldünk, az `~/.arch-params/signing/=default` állománynak a következő egysorost kell tartalmaznia (a cím helyére a kulcs alkotásakor használt címet írjuk):

```
$ echo \
  ´gpg --default-key "<jrh@zork.net>"
  ➥--clearsign´ \
  > ~/.arch-params/signing/\ =default
```

Tükrök

A korábbi szemezgetős példánkban, *Alice B. Hacker* személyes archívumát weben tette elérhetővé. Ez ugyan kényelmes, de ha időnként lekapcsolódunk, problémákat vet fel. Mi van ha *Alice* a noteszgépéről szeretne dolgozni hosszú repülőútjai vagy vonatút közben? Nos, vagy változáskészlet tarlabdákat készít a `tla changes` segítségével, vagy kézzel, egyesével teszi fel noteszgépéről a különféle ágakat a `star-merge` segítségével amikor ismét netközelbe kerül. Szerencsére, az *Arch* lehetővé teszi olyan archívumok készítését is, melyek egyszerűen egy másik archívum tükörképei:

```
$ tla make-archive -ls --mirror-from \
  jrh@zork.net--signed \
  sftp://jrh@zork.net/public_html/arch/
```

A fenti `make-archive` parancsban *J. Random Hacker* egy internet-kiszolgálón hoz létre archívumot saját *public_html* könyvtárában. Miután a tükör archívum elkészült, `jrh@zork.net--signed-MIRROR` néven felkerül a `tla archives` listába. Mostantól egyetlen paranccsal adatot tölthetünk bele:

```
$ tla archive-mirror jrh@zork.net--signed
```

A feltöltő tükrökön kívül, melyek helyi archívum adatokat töltenek távoli rendszerekre, az *Arch* támogatja a lehúzó tükröket is melyekkel távoli források helyi tükörképeit készíthetjük el:

```
$ tla make-archive -ls --mirror \
  lnx-bbc-devel@zork.net--gar \
  /var/tmp/gar-cache
$ tla archive-mirror lnx-bbc-devel@zork.net--gar
```

Ez igen jól jöhet ha szétkapcsolódva dolgozunk, amikor a helyi ág már nem elegendő. A letöltő tükrök (pull mirrors) csak olvasási hozzáférést engedélyeznek a távoli archívum adataihoz amíg nincsenek a hálózatra csatlakozva.

Aláírt tükrök

A `jrh@zork.net--signed-MIRROR` archívum egyik hátránya, hogy maga is külön aláírt archívum. Így aztán *J. Random Hacker* kénytelen minden változáskészletet aláírni amikor az eredeti archívumból a tükörre másolja őket.
Bizonyos esetekben ez kívánatos is lehet. Például, jó, ha a kiadásfelelős külön-külön jóváhagy minden egyes változtatást ami kikerül a nyilvános kiszolgálóra. A legtöbb esetben azonban fontos lenne a változáskészlettel együtt egyszerűen csak átmásolni a már meglévő aláírásokat is. Ezt egy különleges állomány létrehozásával érhetjük el a `tla archive-mirror` parancsot futtató rendszeren:

```
$ echo jrh@zork.net--signed > \
  ~/.arch-params/signing/
  ➥jrh@zork.net--signed-MIRROR
```

Hálózaton kívüli munka (szórakozottaknak kis)

A tükrök rendkívül hasznos eszközök, de természetüknél fogva csak olvashatóak. Az egyetlen mód, amivel a változtatások eljuthatnak a tükörhöz az eredeti archívumon keresztül vezet a `tla archive-mirror` parancs használatával.
Vegyük *Alice* noteszgép tükör felállását. Mialatt az *Amtrak Coast Starlight* kocsijában utazik, előhúzza a noteszgépét és a `tla get` segítségével letölt valamennyi kódot az `abh@zork.net--work` nevű helyi tükörről. Valahol a *Willamette* völgyben ihletet kap és egy figyelemreméltó módosítást készít.
Ha megpróbálná elküldeni a változtatásait, egyfolytában a közvetlen tükörre írás hibaüzenetet kapná, ami azt jelenti, hogy a küldés sikertelen volt. Egyszerű megoldás lehetne,

hogy megvárja, míg elér egy internet elérési pontot, majd az undo és redo parancsokat használja:

```
$ tla undo ,changes-to-mirror
$ cd ~/real-project/
$ tla redo ~/mirror-checkout/,changes-to-mirror/
$ tla commit
```

Ez egészen jól működik, amennyiben a változtatásainknak egy változáskészletnél többre nincs szüksége. Azonban egy hosszabban elhúzódó fejlesztés során új helyi ágat szeretnénk létrehozni.
Miután megérkezett az ***Csendes óceáni*** partra, ***Alice*** felszáll a ***Chicago*** felé induló ***Zephyr***-re. Ez már hosszabb út, és azon kapja magát, hogy a ***bob@entar.net--code*** helyi tükrén található ***foo--stable--2.4.2*** kódon dolgozik. Néhány óra munka után úgy dönt, a változásokat új ágba veszi fel. Először is készít egy új archívumot és ágat a noteszgépén:

```
$ tla make-archive -l abh@zork.net--laptop ~/arch
$ tla my-default-archive abh@zork.net--laptop
$ tla archive-setup foo--laptop-hacks--1.0
```

Következő lépésként felteszi a tükör ágat az új archívumába. A `tla logs` parancsot parancsértelmező fordított aposztrófjai között futtatja így nem kell emlékeznie rá melyik foltozási szinten és verzión dolgozott éppen:

```
$ tla tag `tla logs -r -f | head -n 1` \
  foo--laptop-hacks--1.0
```

Végül, ***Alice*** kényszeríti a kikért másolatot, hogy azt higgye ez az első revízió az új ***laptop-hacks*** ágban:

```
$ tla sync-tree foo--laptop-hacks--1.0--base-0
$ tla set-tree-version foo--laptop-hacks--1.0
```

Ezzel a módszerrel a csak olvasható tükörről lekért másolatát a laptopján lévő írható olvasható archívumba telepítette át.

Ágkészítés Archívum nélkül

Tükör készítése egy hosszabb hálózaton kívüli időszakra ahhoz hasonlít, mint amikor útra csomagolunk: csak azt felejtjük otthon, amire gazán szükségünk lenne. Elég kiábrándító tud lenni, amikor az ember a notebookját hegyi lakának lámpa-dugójába bedugva ráébred, hogy a kimásolt változat bizonyos létfontosságú részei valamilyen ***HTTP*** archívumból származnának.
Szerencsére, ugyanezeket a technikákat felhasználhatjuk a kikért másolat új ágba mozgatásához akkor is, éppen ha nem tudjuk elérni a régi archívumot.
Alice közben egy ***Chicagoi Internet*** szalonban üldögélve lekérte a ***bar*** nevű projekt másolatát. ***Kaliforniába*** visszautazva, úgy dönt dolgozik egy kicsit a kódon. Újabb óra bámulatos erőfeszítés után ismét úgy találja, hogy ideje új ágat készíteni a munkájához.
Minthogy az eredeti forrás elérhetetlen, az ág felcímkézése lehetetlen. Szerencsére, a kikért fában a változásnapló és a történet információk nagy része megtalálható, így ***Alice*** ideiglenesen elmenti változtatásait a `tla undo` paranccsal, majd a kikért másolatra rákényszeríti saját új ágát:

```
$ tla archive-setup bar--train-ride--1.0
$ tla set-tree-version bar--train-ride--1.0
$ tla add-log-version bar--train-ride--1.0
$ tla import
```

Miután ezzel megvan, ***Alice*** lefuttatja a `tla redo` majd a `tla commit` utasítást. A ***Chicagoban*** leszedett változat neve most a `bar--train-ride--1.0--base-0`, módosított változata pedig a `bar--train-ride--1.0--patch-1` nevet viseli.
Bár a módszer nem tökéletes, a `star-merge` parancsot továbbra is gond nélkül használhatjuk az eredeti ágba és ágból való adatmozgatásra. Ha ***Alice*** úgy látja, hogy a bar projekt kellő mértékben érintett, valószínűleg összeolvasztaná a vezető archívummal és elkészítené a megfelelő ágat, amint ismét internetközelbe kerül.

Finomhangolásról legközelebb

Most már tudjuk, hogyan kell terjesztenünk az archívumainkat az Interneten és hogyan dolgozzunk távolról az ***Arch***-al. Még akkor is akad egy két trükk a tarsolyunkban, ha esetleg hibát követnénk el.
A sorozat következő, egyben utolsó részében bemutatjuk, hogyan tartsuk karban a központosított hivatalos archívumainkat az ***Arch*** megoszló munkarendszerének minden előnyét megtartva. Megtanulunk néhány archívumokkal kapcsolatos parancsfájlírási trükköt amelyekkel jelentéseket készíthetünk a fejlesztési aktivitásról, valamint létrehozunk egy tesztkörnyezetet.

Linux Journal 2004. december, 128. szám

- Mindenki követ el hibákat, nem is ritkán, így aztán elég fontos hogy eszközeink rendelkezzenek valamilyen barátságos helyreállítási lehetőséggel.
- Az Arch egyik nagy előnye elődjéhez a CVS-hez képest, hogy támogatja a változtatás-készletek létrehozását és kezelését.
- Az ***Arch*** segítségével ***Alice*** és ***Bob*** képesek voltak egymás egyedi archívumán és a különbségeken dolgozni, miközben semmi különlegeset nem használtak csak az ***Apache*** és az ***OpenSSH*** kiszolgálókat.

Nick Moffitt
San Francisco öbölvidékén élő hivatásos linuxos. Ő a GNU/Linux LNX-BBC Bootable Business Card változatának fejlesztőmérnöke és a GAR fordítórendszer szerzője. Amikor éppen nem programokat bütyköl, a tömegközlekedés történelmét tanulmányozza.