

# Работаем с данными Active Directory из скриптов

## Часть первая. Применение Bourne Shell

**Рашид Ачилов**

**В корпоративной среде при разработке скриптов рано или поздно возникает вопрос: «А неплохо было бы иметь базу всех сотрудников с именами, должностями, адресами и номерами ICQ...», что зачастую приводит к самостоятельным разработкам разной степени удачливости. Но есть и более простой способ. Такая база уже есть. Ее за нас придумала Microsoft. Это Active Directory. Нужно только уметь с ней общаться.**

### **Просто LDAP, и ничего особенного**

В статье не будет рассматриваться, что такое Active Directory и как она работает – на эту тему написано множество книг, например [1, 2], а также описание принципов работы Active Directory (договоримся в дальнейшем

называть ее AD) в серьезных справочниках для администраторов Windows, например [3]. Мы обсудим здесь только те особенности, которые пригодятся нам в дальнейшем.

Для тестирования всюду будет использоваться Windows 2003 Server с именем AD shelton.net, коротким име-

нем (pre-Windows 2000) «CATS», именем самого компьютера BIGCAT.

Человек, никогда не видевший AD, получив сетевой доступ к серверу AD, непременно воскликнет: «Так ведь это же сервер LDAP!». И будет прав. Действительно, со стороны сети сервер AD выглядит как сервер LDAP.

Вот какие открытые порты можно обнаружить на нем:

```
Starting Nmap 4.62 ( http://nmap.org ) at 2008-08-01 01:45 NOVST
Interesting ports on bigcat (192.168.50.1):
Not shown: 1699 closed ports
PORT      STATE SERVICE
42/tcp    open  nameserver
53/tcp    open  domain
88/tcp    open  kerberos-sec
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
593/tcp   open  http-rpc-epmap
636/tcp   open  ldaps
1026/tcp  open  LSA-or-nterm
1027/tcp  open  IIS
1067/tcp  open  instl_boots
3268/tcp  open  globalcatLDAP
3269/tcp  open  globalcatLDAPssl
3389/tcp  open  ms-term-serv
```

Это типичный набор сервисов для контроллера домена (domain controller, dc). Он может слегка варьироваться, но сервис, который нас интересует – ldap (порт 389), – обязан присутствовать. Неизвестно, почему Microsoft решила не изобретать свой велосипед, как поступает обычно, а использовало хорошо известный и хорошо документированный LDAP. Зато над созданием схемы LDAP она «поработала» изрядно – скажем, на схему inetOrgPerson ее схема непохожа совершенно – типичный объект AD «пользователь» с точки зрения ldapsearch выглядит вот таким образом:

```
dn: CN=LDAP Reader,CN=Users,DC=shelton,DC=net
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: LDAP Reader
sn: Reader
givenName: LDAP
distinguishedName: CN=LDAP Reader,CN=Users,DC=shelton,DC=net
instanceType: 4
whenCreated: 20080817091820.0Z
whenChanged: 20080817091820.0Z
displayName: LDAP Reader
uSNCreated: 954394
uSNChanged: 954399
name: LDAP Reader
objectGUID:: S2M6nZo2XkCfS2jjHt4XBg==
userAccountControl: 66048
badPwdCount: 0
codePage: 0
countryCode: 0
badPasswordTime: 128634384464218750
lastLogoff: 0
lastLogon: 128634384682968750
pwdLastSet: 128634383003906250
primaryGroupID: 513
objectSid:: AQuAAAAAAUVAAtInjRe4dmI0lvJgpVwQAAA==
accountExpires: 9223372036854775807
logonCount: 0
sAMAccountName: ldapread
sAMAccountType: 805306368
userPrincipalName: ldapread@shelton.net
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=shelton,DC=net
```

Это, разумеется, далеко не полный перечень полей объекта, а только те, что создаются при создании учетной записи пользователя с минимальным вводом данных о нем.

Ну и раз мы условились считать AD «просто LDAP», то, естественно, к ней будут применимы все средства тестирования для работы с LDAP, например, ldapsearch, программа из комплекта OpenLDAP Toolkit, а также ldapbrowser –

программа, написанная на Java (Swing), для работы с LDAP в графическом режиме (см. **рисунок**).

Что касается ldapbrowser, то с ним практически сразу все ясно (кроме настроек подключения, которые станут ясными сами по себе после описания ldapsearch). А вот о ldapsearch хотелось бы поговорить особо.

Ldapsearch – это программа, чрезвычайно важная для администратора LDAP, примерно такая же, как ping и traceroute для администратора сетевого. Она позволяет получить, а после несложных манипуляций и использовать практически любые данные, что есть в LDAP. Но...

У нее собственный язык запросов. Он не то что бы является сильно сложным, но повергает в изумление любого, кто сталкивается с ним в первый раз. Он построен в соответствии с принципом «обратной польской записи», как в старинных программируемых калькуляторах, когда сначала задается знак операции, а потом перечисляются операнды. Все операнды разделяются отдельными скобками, из-за этого выражения приобретают невероятную громоздкость. Поскольку это язык запросов, то в нем присутствуют только логические операции – И (&), ИЛИ (!), НЕ (!). Не путайте ИЛИ и НЕ, несмотря на то, что они сходны по начертанию! Например:

- **(description=\*)** – любой объект, у которого определено поле description. При этом совершенно не важно будет, сколько и каких в поле символов – один символ «?» точно так же попадет под этот фильтр, как и «длинная-длинная строка».
- **(&(objectClass=user)(mail=\*))** – любой объект, у которого поле objectClass равно user и определено поле mail.
- **(&(objectClass=user)(!(mobile=100\*)(mobile=200\*)))** – любой объект класса user, у которого поле mobile начинается с цифр 100 или 200.
- **(&(sAMAccountName=\*)(homeMTA=\*)(!(title=\*)(department=\*)))))** – любой объект, у которого определены поля sAMAccountName и homeMTA и не определены поля title или department. Обратите внимание на расположение знаков логических операций и баланс скобок в последнем примере.
- **(&(&(mailnickname=\*)(!(objectCategory=person)(objectClass=user)(!(homeMDB=\*)))(!(msExchHomeServerName=\*))(&(objectCategory=person)(objectClass=user)(!(homeMDB=\*)(msExchHomeServerName=\*))(&(objectCategory=person)(objectClass=contact))(objectCategory=group)(objectCategory=publicFolder))))))** – этот поистине чудовищный запрос используется в KAddressbook для получения глобальной адресной книги из LDAP. Вы можете попрактиковаться в его разборе самостоятельно. Создавал его не я, мне его посоветовали в рассылке, посвященной KDE PIM. Тот, кто его создавал, явно не знал ничего про структуру AD, потому что то же самое можно добиться запросом **(&(sAMAccountName=\*)(!(sAMAccountType=805306368)(sAMAccountType=268435456))(mail=\*))**. В приводимых числах в sAMAccountType нет ничего магического – это просто 0x30000000 и 0x10000000 в десятичной форме.

Более подробно язык запросов описан в [5].

С языком запросов немного разобрались. Остается сле-



дующий вопрос – как подключиться к LDAP? Здесь надо учесть несколько вещей.

Во-первых, в Windows Server отсутствует так называемый anonymous bind, то есть возможность подключения к LDAP без предъявления какого-либо имени и пароля. Это, конечно, можно включить, но по умолчанию это выключено. Поэтому перед тем, как начать работать с LDAP через ldapsearch или разрабатывать скрипты, необходимо создать отдельную учетную запись, отобрать у нее все права, какие только можно, установить достаточно простой пароль, не содержащий символов, которые при передаче через шелл могут быть интерпретированы (прямые и обратные слэши, амперсанд, знак вопроса и т. д.). Я обычно использую пароль «qwerty{123}» – почему-то системой Windows 2003 Server он считается достаточно сложным и имя пользователя ldapread. Для ldapsearch (и далее повсюду) в параметре, задающем так называемый bind dn, указывается строка <имя\_пользователя>@<ДОМЕН>, причем с указанным соблюдением регистра, например, ldapread@SHELTON.NET. <ДОМЕН> – это имя AD, переведенное в верхний регистр.

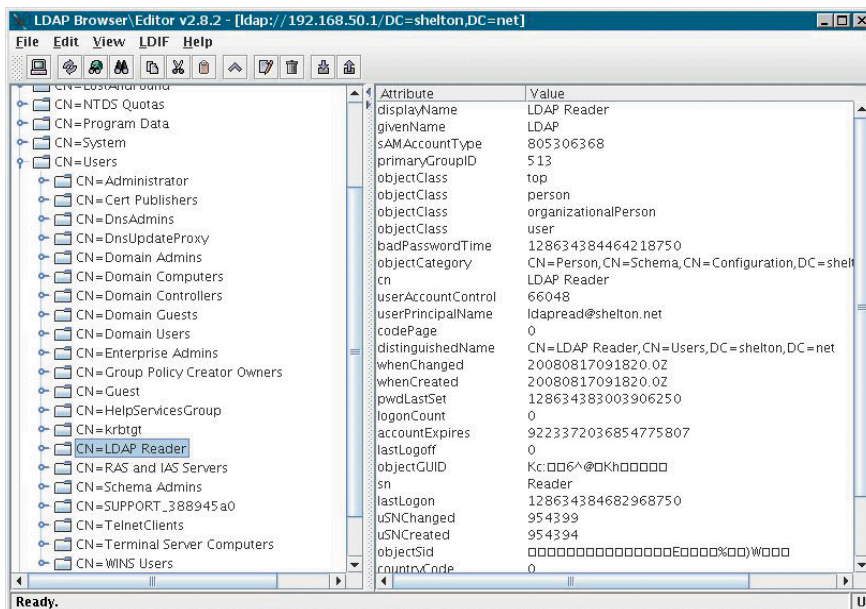
Во-вторых, что указывать в качестве корня для подключения (base dn)? Обычно base dn формируется следующим образом: если AD называется, допустим, shelton.net, то соответствующее значение base dn будет dc=shelton,dc=net. Если же оно называется, например, kitten.cats.shelton.net (такие длинные имена возможны в больших инсталляциях), то соответствующее значение base dn будет dc=kitten,dc=cats,dc=shelton,dc=net. Разумеется, можно при подключении указать не корень AD, а какую-либо его ветку, например, CN=Computers,DC=shelton,DC=net, но в таком случае поиск никогда не поднимется выше ее границы. Проблема же состоит в том, что логически правильным является создание организационных единиц (OU – Organizational Unit), допустим, соответствующих разным филиалам непосредственно в корне AD, поскольку эти ou будут содержать объекты разных типов – пользователей, компьютеры, группы, другие OU. Поэтому для просмотра всей AD лучше указывать корень.

В третьих, как в AD хранится информация, заносимая не ASCII символами, то есть на национальных (и на русском в том числе) языках? Хранится она в виде текста в кодировке UTF-8, преобразованного с помощью Base64 в набор символов ASCII. Подробности кодирования изложены в [6]. В том, что там именно Base64, мы вскоре убедимся.

Теперь пора попробовать ldapsearch в деле.

```
# ldapsearch -D ldapread@SHELTON.NET -w qwerty{123} -L
-L -LLL -h 192.168.50.1 -b dc=shelton,dc=net -P 3 -L
-a always (sAMAccountName=ld*) distinguishedName sAMAccountType
```

```
dn: CN=LDAP Reader,CN=Users,DC=shelton,DC=net
sAMAccountType: 805306368
```



Ldapbrowser – программа работы с LDAP

Использованные ключи имеют следующее значение:

- -D – задает binddn;
- -w – пароль для подключения;
- -LLL – обеспечивает максимальное упрощение выводимой информации;
- -h – адрес сервера;
- -b – задает basedn;
- -P – версия протокола LDAP;
- -a – описан в man ldapsearch.

Далее идет фильтр отбора записей (простейший) и перечень атрибутов, которые нужно вывести. Если не указать ничего, будут выведены все атрибуты. В данном случае мы запросили атрибуты distinguishedName и sAMAccountType. Обратите внимание, что атрибуты, которые необходимо вывести, перечисляются через пробел, а не через запятые.

Теперь добавляем в информацию о пользователе (средствами Microsoft) поле title, равное «Читатель», и запрашиваем его через ldapsearch:

```
# ldapsearch -D ldapread@SHELTON.NET -w qwerty{123} -L
-L -LLL -h 192.168.50.1 -b dc=shelton,dc=net -P 3 -L
-a always (sAMAccountName=ld*) title sAMAccountType
```

```
dn: CN=LDAP Reader,CN=Users,DC=shelton,DC=net
title:: OKfQuNGC0LDRgtC10LvRjA==
sAMAccountType: 805306368
```

Здесь атрибут title закодирован согласно [6], что легко узнается по двум знакам «==». Проверяем:

```
# echo OKfQuNGC0LDRgtC10LvRjA== | mmencode -u | iconv -L
-f utf-8 -t koi8-r
```

Читатель

Как мы видим, [6] не обманывает нас – это действительно строка UTF-8, закодированная через Base64. Этот прием – перекодирование через mmencode и преобразование через iconv – это уже практически готовая часть скрипта, к рассмотрению которого мы сейчас и приступим.

## Работаем из Bourne Shell

Разумеется, решать «задачу ради задачи» скучно. Поэтому возьмем вполне конкретный пример – для использования программой Sarg сформировать так называемый usertab-файл, который задает соответствие логинов, использованных в процессе аутентификации в Squid и их имен, так как они будут выведены в отчете, так что получается вроде:

ivanov-iv	Иванов Иван
petrov-pp	Петров Петр
root	Администратор

Последовательность действий будет практически такая же, как мы только что проводили вручную – подключаемся к LDAP с соответствующим фильтром и списком полей, получаем вывод, преобразовываем его и сохраняем в виде файла. Нас интересуют только поля displayName и sAMAccountName. Поскольку это демонстрационный скрипт, а не полная версия (полная версия содержит больше элементов и выполняется в соответствующем окружении), то некоторые части его будут опущены – они в принципе уже не раз приводились и могут быть взяты из любого другого скрипта с [4]. Рассматриваться будут только те части, которые непосредственно отвечают за выполнение интересующих нас действий. Так, например, не будут рассматриваться функции usage() и safe\_logger().

Данный скрипт будет использовать конфигурационный файл типового формата, в котором собраны настройки подключения к LDAP:

```
ldap_server=192.168.50.1
ldap_basedn="dc=shelton,dc=net"
ldap_binddn=ldapread@SHELTON.NET
ldap_password="cXdlcnR5YXNkZgo 1"
ldap_common_filter="(sAMAccountName=*)"
(sAMAccountType=805306368)(telephoneNumber=*)"
etcdir=/usr/local/etc
sargdir=sarg2
sargfile=sargusers
```

Здесь все должно быть понятно. Кроме пароля. Для пароля применена такая странная запись по следующим причинам. Во-первых, то, что хранить пароли в открытом виде – зло, знают все. Но в то же время трудно найти обратимую функцию для того, чтобы надежно зашифровать пароль, и потом столько же надежно его расшифровать, не прибегая к помощи PGP. Поэтому был выбран компромисс – простейшее преобразование формата, защищающее от случайного взгляда, и не более. Это именно преобразование, а не шифр. Точнее говоря, это Base64, у которого отброшены заполнители и вместо них проставлено их число, чтобы слегка запутать. Разворачивается этот пароль вот такой несложной процедурой:

```
# Обратное преобразование строки пароля
# Вход: $1 enconvd (строка) - пароль из файла конфигурации
# Выход: _passwd - пароль в текстовом виде
demux_passwd()
{
    local ideconv iadd ifill
    ifill=""
    ideconv=$1
    if [ ! $2 -eq 0 ]; then
        iadd=$2
        while [ $iadd -gt 0 ]; do
            ideconv=$(ideconv$ifill)
            iadd=$((iadd-1))
        done
    fi
    _passwd=$(echo $ideconv | $util_mmencode -u)
}
```

```
done
fi
_passwd=$(echo $ideconv | $util_mmencode -u)
}
```

Пояснять тут особо нечего – если второй параметр не нуль, то циклом дописать в конец строки столько знаков заполнения, сколько передано во втором параметре, и декодировать строку.

Основной принцип работы скрипта следующий: после вводной части (загрузки конфигурационного файла, разбора командной строки, поиска необходимых программ) скрипт «расшифровывает» пароль, создает временный каталог для работы, потом запрашивает из LDAP логины и имена пользователей и сохраняет их во временных файлах (процедуры описаны ниже и довольно подробно):

```
safe_logger "LDAPQuery ver. $revisionNumber started"
# Обратное преобразование пароля
demux_passwd $ldap_password
ldap_pwd=$_passwd
# Создать временный каталог, если еще не создан,
# и перейти в него
make_tmpdir
# Выполнить поиск в LDAP и сохранить результат
ldap_data_query sAMAccountName $tmpdir/users.ldif
# Преобразовать данные в читаемый вид
ldap_data_deconvert $tmpdir/users.ldif $tmpdir/users.list
# Выполнить поиск в LDAP и сохранить результат
ldap_data_query displayName $tmpdir/names.ldif
# Преобразовать данные в читаемый вид
ldap_data_deconvert $tmpdir/names.ldif $tmpdir/names.list
```

Далее выполняется прием, который я применяю очень часто – два вложенных цикла сравнения двух списков. Первый список – это список полученных из LDAP логинов, второй – список логинов пользователей, уже присутствующих в файле. Он получается тривиальной командой на awk. Суть его в том, что оба списка считываются в переменные, берется один элемент «внешнего» списка и сравнивается со всеми по очереди элементами «внутреннего» списка. Когда находится совпадение, выполняются необходимые действия, потом внешний цикл переходит к следующему элементу (в Bourne Shell такое возможно – указать оператору break или continue число уровней вложенности, на которые он одновременно воздействует).

Во внешнем цикле, идущем по списку логинов из LDAP, обнуляется счетчик позиции, который будет отмечать позицию имени в списке имен. Внутренний же цикл идет по списку пользователей, уже присутствующих в файле. Если логин из LDAP уже присутствует в файле, то незачем его искать дальше, и переходим к следующему логину. Если же отсутствует, то берется имя пользователя из списка имен, отступив с помощью tail на значение счетчика позиции и взяв оттуда одну строку, после чего формируется строка вывода и выводится в список пользователей Sarg с дозаписью:

```
# Внешний цикл по списку пользователей из LDAP
for _onldapuser in $ldapusers
do
    togo=0
    # Внутренний цикл по списку пользователей Sarg
    for _onesarguser in $sargusers
    do
        # Если пользователь найден, то мы прерываем
        # и внутренний, и внешний циклы.
        # Переходим к следующей записи внешнего цикла
        if [ $_onldapuser = $_onesarguser ]; then
            continue 2
        fi
    done
done
```

```

fi
done
# Если мы находимся здесь, значит, пользователь LDAP
# не найден в списке пользователей Sarg. Мы записываем
# в список пользователей Sarg строку с логином
# из names.list и именем из names.list
_oneusername='cat $tempdir/names.list | \
tail -n +$ togo | head -n 1'
echo -e "$ oneldapuser \t$ _oneusername\n" >> \
$etcdir/$sargdir/$sarglist
_togo=$(( $ togo+1))
_added=$(( $ _added+1))
done

```

Наибольший интерес, конечно, представляют процедуры `ldap_data_query()` и `ldap_data_deconvert()`.

`ldap_data_query()` – достаточно простая процедура, которая всего лишь вызывает `ldapsearch`, передавая ему все необходимые параметры из конфигурационного файла и списка переданных параметров. Единственное, что здесь делается до вызова `ldapsearch`, – это замена запятых в списке запрашиваемых полей на пробелы:

```

# Запросить некоторые данные у LDAP-сервера
# Вход: $1 fieldname (строка) - Имя атрибута(ов),
# запрашиваемых в AD
# $2 outfile (строка) - Имя файла для сохранения
# полученных данных
# Выход: none
ldap_data_query()
{
    local _ldapfield _outfile
    _ldapfield='echo $1 | sed -e "s,:,:g"'
    _outfile=$2
    # Выполнить запрос к LDAP и сохранить результат
    $util ldapsearch -T $tempdir -D $ldap_binddn \
        -w $ ldap_pwd -LLL -h $ldap_server \
        -b $ldap_basedn -P 3 -a always \
        $ldap_common_filter $_ldapfield > $_outfile
    # Проверить код возврата
    status=$?
    if [ $status -ne 0 ]; then
        safe_logger "Unable to complete LDAP request to get \
        $_ldapfield from AD"
        exit
    fi
}

```

А вот процедура `ldap_data_deconvert()` гораздо интереснее. Предварительно рассмотрим вспомогательные процедуры `unspace()` и `make_tempdir()`.

```

# Удалить пробелы и строки комментариев из файла
# Вход: $1 source (строка) - Имя исходного файла
# $2 destination (строка) - Имя файла результата
# Выход: none
unspace()
{
    cat $1 | grep -v ^# | sed -e "s,:,:g" > $2
}

```

Процедура `unspace()` удаляет из файла, заданного первым параметром, все строки, начинающиеся со знака «#», а также удаляет все пробелы. В итоге на выходе будут только строки вывода `ldapsearch`:

```

# Создать временный каталог, если он еще не был создан,
# и перейти в него
# Вход: none
# Выход: tempdir (строка) - имя временного каталога
make_tempdir()
{
    if [ ${tempdir} -eq 0 ]; then
        tempdir='mktemp -d /tmp/adphones.XXXXXX'
    fi
    cd $tempdir
}

```

`make_tempdir()` исключительно проста – она создает временный каталог с уникальным именем, если такового еще не было создано. Сделано это для удобства использования данного скрипта из других скриптов, чтобы избежать гонок (race condition).

Наиболее сложными для понимания являются две основные процедуры – `ldap_data_deconvert()` и `write_when_not_dn()`. В чем их смысл?

Дело в том, что `ldapsearch` в обязательном порядке выводит строку `dn` для того объекта, поля которого печатаются. Эту строку необходимо пропустить. Но здесь мы имеем еще одну проблему: если в `dn` есть символы национальных алфавитов, то оно будет представлено `base64`-кодом. Поскольку `dn` может быть очень длинным, то кодировка его зачастую значительно превышает по длине 80 символов, а у `ldapsearch` есть еще одна «особенность» – после 80 позиций текста обязательно ставится символ перевода строки, так что просто разобрать по строкам не получается. И выбрасывать переводы строк нельзя – тогда пропадет разбиение между атрибутами. Поэтому и пришлось сделать несколько нетривиальных ходов. Для описания `ldap_data_deconvert()` скажем, что `write_when_not_dn()` выводит данные, требующие преобразования в один файл, не требующие – в другой и пропускает поле `dn` при чтении. Кроме того, есть еще одна очень важная для нас особенность `ldapsearch` – если атрибут содержит данные в `base64`, то значение отделяется от имени атрибута двумя символами двоеточия вместо одного, то есть если есть второе поле – это данные, которые нет необходимости перекодировать, если его нет, но есть третье – это данные, которые необходимо перекодировать.

Собственно же `ldap_data_deconvert()` выглядит следующим образом:

```

# Обратное преобразование данных, прочитанных из AD
# Вход: $1 source (строка) - имя файла с исходными
# данными
# $2 dest (строка) - имя файла для записи
# выходных данных
# Выход: none
ldap_data_deconvert()
{
    local _tempsfx _outfile _tempstr _enconvded _skip
    # Из строки типа /path/to/filename.ext мы выбираем
    # только имя файла. Первая команда отбрасывает путь
    # (шаблон - наибольший префикс). Вторая команда
    # отбрасывает расширение (шаблон - наименьший суффикс)
    _tempsfx=${1##*/}
    _tempsfx=${_tempsfx%.*}
    _outfile=$2
    # Удаляем из файла все пробелы и строки комментариев
    unspace $1 stripped.$_tempsfx
    # Готовимся разбивать по строкам, для чего используем
    # глобал $newline, равный переводу строки
    saveifs=$IFS
    IFS=$newline
    _skip=0
    _tempstr=""
    _enconvded=0
    convded='cat stripped.$_tempsfx'
    for one in $convded
    do
        # Внутри строки разбиваем по символу двоеточия
        IFS=:
        set $one
        # Если $2 установлен, это значит, что значение в коде
        # ASCII и может быть записано в выходной файл
        # напрямую, если это не dn
        write_when_not_dn "$2" $1 1
        # Если $3 установлен, значит, что значение в коде
        # base64 и должно быть преобразовано перед записью,
        # если это не dn
        write_when_not_dn "$3" $1 0
    done
}

```

```
# Если же одна переменная - это промежуточная строка,
# и она должна быть присоединена к значению буфера
if [ ${#2} -eq 0 ] && [ ${#3} -eq 0 ] && [
    [ $ skip -eq 0 ]; then
    _tempstr=$_tempstr$1
fi
IFS=$newline
done
# Если данные не требуют преобразования - выводим
# в один файл, если требуют - в другой, потому что
# mmencode пытается преобразовать все данные,
# переданные ей
if [ $_enconvd -eq 1 ]; then
    echo $_tempstr >> unconvd.data
else
    echo $_tempstr >> convd.data
fi
# Сортируем данные в обоих файлах и удаляем
# дублирующие строки
cat unconvd.data | sort -df | uniq > $_outfile
cat convd.data | sort -df | uniq > deconvd.data
# Построчно перекодировываем файл
deconvd='cat deconvd.data'
for one in $deconvd
do
    echo $one | $util mmencode -u | $util iconv -f
    -f utf-8 -t koi8-r >> $_outfile
    printf "\n" >> $_outfile
done
IFS=$saveifs
}
```

Первые три строки после объявления переменных – пример того, как можно отбросить путь (еще для этого используется \$(basename \$1)) и расширение файла, оставив только его имя. Затем отбрасываем в файле все комментарии и пустые строки (unpspace()) и начинаем его построчную разборку. Поскольку в выводе ldapsearch имя поля отделяется от данных двоеточием, каждую строку разбиваем по этому символу.

Если после разбора есть второе поле, то данные не нужно преобразовывать, достаточно проверить, не dn ли это (write\_when\_not\_dn()). Если же после разбора есть третье поле (второе при этом не проверяется), то значит, это base64-данные, и они должны быть преобразованы, опять же, если это не dn (write\_when\_not\_dn()). Если же нет ни второго, ни третьего полей – это промежуточная строка закодированных данных, и она должна быть присоединена к буферу, в которые помещается недописанная строка. Это продолжается до конца входного файла.

Если были обнаружены закодированные данные, то сбросить последнюю строку данных в файл с данными для декодирования, иначе в файл, который не будет декодирован. Дело в том, что mmencode не проверяет переданный ей текст, а пытается перекодировать его, независимо от содержимого.

После этого отсортировать файлы, отобрать уникальный текст (зачем перекодировать что-то более одного раза?) и декодировать файл с закодированным текстом. Декодирование идет по одной строчке с последующим преобразованием в KOI8 (это можно убрать, кому не нужно) и выводом символа перевода строки после каждой строки.

Последней интересной процедурой является write\_when\_not\_dn(). Она не очень большая, но весьма запутанная, потому что постоянно используются внешние переменные:


```
# Сформировать файл с данными для преобразования, если оно
# требуется, иначе с данными, не требующими преобразования,
# в том случае, если это не dn
# Вход: $1 line (строка) - данные из исходного файла
```

```
# $2 attribute (строка) - имя атрибута
# $3 enconvd (digit) - 1 значит, что данные
# не требуют преобразования
# Выход: none
write_when_not_dn()
{
    # Если установлен $1, значит, это первая строка данных,
    # и нужно записать предыдущее содержимое
    # промежуточного буфера
    [ ${#1} -eq 0 ] && return
    # Если промежуточный буфер содержит накопленные данные,
    # их нужно записать
    if [ ${#_tempstr} -ne 0 ] && [ $ skip -ne 1 ]; then
        # Если данные не требуют преобразования, записать
        # в один файл, если требуют - в другой, потому что
        # mmencode преобразовывает все
        if [ $_enconvd -eq 1 ]; then
            echo $_tempstr >> unconvd.data
            _enconvd=0
        else
            echo $_tempstr >> convd.data
        fi
    fi
    # Если атрибут - dn, мы пропускаем эту и следующие строки
    # до тех пор, пока не начнется
    # новый атрибут
    if [ $2 = "dn" ]; then
        # Пропускать следующие строки
        _skip=1
        return
    else
        # Использовать следующие строки и накапливать данные
        # во временном буфере
        _skip=0
        _tempstr=$1
        # Отметим, если данные не требуют преобразования
        if [ $3 -eq 1 ]; then
            _enconvd=1
        fi
    fi
}
```

Если первый параметр не задан (а такое сплошь и рядом, когда разбираются закодированные данные), то сразу выйти, ничего не делать. Если существуют данные во временном буфере, сброшенные туда при предыдущем вызове, то их нужно записать в соответствующий файл.

Если атрибут – dn, то пропустить данные, переданные при вызове, и все следующие строки, пока не будет снова обнаружено два поля в строке. Иначе – поместить данные во временный буфер, и, если они не требуют преобразования, отметить это.

## Заключение

Язык Bourne Shell не очень-то годится для таких вещей, как обработка данных, полученных из AD, честно говоря. Приходится пускаться на различные ухищрения, и все равно работа получается с ограничениями – например, мне так и не удалось сделать одновременный запрос нескольких полей. Но для решения несложных задач типа приведенной его вполне можно использовать. 

1. С. Реймер, М. Малкер. Active Directory для Windows 2003 Server. Справочник администратора. Пер. с англ. – М.: «СП-ЭКОМ», 2004 – 512 с.
2. Active Directory, 2nd Edition. Robbie Allen, Alistair G. Lowe-Norris. O'Reilly, 2003. ISBN 0-596-00466-4.
3. Ч. Расселл, Ш. Кроуфорд, Дж. Деренд. Microsoft Windows 2003 Server. Справочник администратора/Пер. с англ. – М.: Издательство «СП ЭКОМ», 2004 – 1392 с.: ил. ISBN 5-9570-0016-7.
4. <http://openoffice.mirahost.ru> – Сайт, где выложен скрипт.
5. <http://www.ietf.org/rfc/rfc2254.txt>.
6. <http://www.ietf.org/rfc/rfc2849.txt>.