# Введение

# 1.Теоретическая глава

Windows PowerShell — новая командная строка и язык сценариев от компании Microsoft. PowerShell является компонентом Windows Server 2008 (надо только выбрать его в Server Manager) и доступна для загрузки со странички www.microsoft.com/powershell для Windows XP, Windows Server 2003 и Windows Vista.

Если вы не знакомы с Windows PowerShell, то рекомендуем вам сначала прочитать статью «Windows PowerShell. Коротко о главном» в КомпьютерПресс № 11’2007. В данной публикации мы ограничимся лишь кратким повторением основ и сразу перейдем к главной теме статьи.

Итак, команды PowerShell называются командлетами (cmdlet) и состоят из глагола (например, get, set, new, remove, move, connect) и существительного в единственном числе, описывающего объект действия. Между ними ставится дефис. Получается что-то вроде: get-process, stop-service и т.п.

Команды, как правило, связываются конвейером, обозначаемым вертикальной чертой (|). Этот знак означает, что вся коллекция объектов из предыдущей команды передается на вход следующей.

Такая объектная ориентированность очень удобна, поскольку позволяет легко оперировать объектами и связывать команды вместе. В этой статье мы расскажем, как подобный подход облегчает управление корпоративной директорией на базе Active Directory.

## 1.1 Способы работы с Active Directory

Директория Active Directory является основой корпоративных сетей на базе Windows Server 2000, 2003 и 2008. Именно там хранятся все учетные записи пользователей, информация о группах, компьютерах сети, ящиках электронной почты и многом другом.

Всем этим богатством надо управлять, для чего предназначен соответствующий инструментарий, входящий в состав Windows Server, но именно PowerShell позволяет легко автоматизировать массовые действия, направленные на большое количество объектов.

Существует три основных способа работы с Active Directory в Windows PowerShell:

* с помощью интерфейса Active Directory Service Interfaces (ADSI) — этот способ является наиболее сложным, но работает в любой установке PowerShell и не требует дополнительных модулей. Он также наиболее близок к способу управления, который использовался в языке сценариев VBScript;
* с помощью провайдера Active Directory, входящего в расширения PowerShell, — этот способ позволяет подключить директорию в виде диска на вашем компьютере и перемещаться по ней с помощью соответствующих команд: dir, cd и т.д. Данный способ требует установки дополнительного модуля с сайта codeplex;
* с помощью командлетов управления Active Directory — это наиболее удобный способ манипулирования объектами директории, но он тоже требует дополнительной инсталляции соответствующих модулей.

## 1.2 ADSI

Active Directory Service Interfaces (ADSI) хорошо знаком всем, кто пытался писать сценарии на языке VBScript. В PowerShell этот интерфейс реализован с помощью так называемого адаптера. Указав в квадратных скобках название адаптера (ADSI) и путь к объекту в директории на языке LDAP-запроса (Lightweight Directory Access Protocol — протокол работы с директориями, который поддерживает и AD), мы получаем доступ к объекту из директории и можем дальше вызывать его методы.

Например, подсоединимся к одному из контейнеров директории и создадим в нем новую пользовательскую учетную запись.

$objOU = [ADSI]”LDAP://mydc:389/ou=CTO,dc=Employees,dc=testdomain,dc=local”

Итак, теперь у нас переменная $objOU содержит информацию о контейнере (имена переменных в PowerShell начинаются со значка доллара).

Вызовем метод *Create* и создадим в контейнере нового пользователя:

$objUser = $objOU.Create(“user”, “cn=Dmitry Sotnikov”)

Теперь мы можем устанавливать различные атрибуты:

$objUser.Put(«sAMAccountName”, «dsotnikov”)

И наконец, укажем директории, что эти изменения надо применить:

$objUser.SetInfo()

Преимуществами использования адаптера ADSI являются:

* его наличие в любой поставке PowerShell. Если у вас установлен PowerShell и есть директория, с которой вам надо работать, — вы имеете все, что вам надо;
* применение подхода, близкого к VBScript. Если у вас богатый опыт работы с директорией на языке сценариев VBScript или в приложениях .NET, вы сможете уверенно себя чувствовать, используя этот подход.

К сожалению, у метода есть и недостатки:

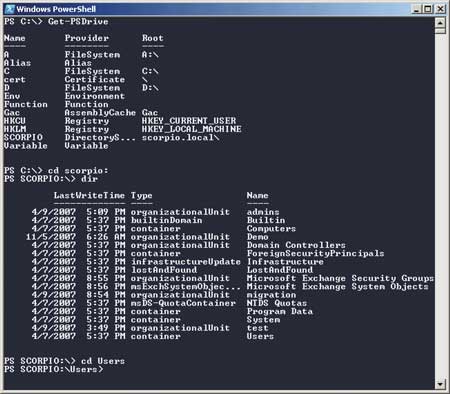
* сложность — это самый сложный способ работы с директорией. Писать путь к объекту в виде запроса LDAP нетривиально. Для любой работы с атрибутами требуется указание их внутренних имен, а значит, надо помнить, что атрибут, обозначающий город пользователя, называется не «City», а «l» и т.д.;
* громоздкость — как видно из примера, простейшая операция создания одной учетной записи занимает как минимум четыре строчки, включая служебные операции подсоединения к контейнеру и применения изменений. Таким образом, даже относительно простые операции становятся похожи на сложные сценарии.

## 1.3 Провайдер AD

PowerShell позволяет представлять различные системы в виде дополнительных дисков компьютера с помощью так называемых провайдеров. Например, в состав поставки PowerShell входит провайдер реестра и мы можем перемещаться по реестру с помощью знакомых и любимых всеми нами команд cd и dir (для любителей UNIX команда ls тоже поддерживается).

Провайдера Active Directory в составе PowerShell нет, но его можно установить, зайдя на сайт проекта расширений PowerShell — PowerShell Community Extensions: <http://www.codeplex.com/PowerShellCX>.

Это проект с открытым кодом, который добавляет большое количество команд в систему PowerShell, а кроме того, устанавливает провайдера AD.

**

*Использование провайдера Active Directory*

После установки расширений, набрав Get-PSDrive, мы видим, что к прежним дискам добавился диск текущей активной директории.

Теперь мы можем зайти в эту директорию, набрав cd и указав имя домена, а в любом контейнере использовать команду dir, чтобы увидеть его содержимое.

С объектами дальше можно работать так же, как это делалось с применением адаптера ADSI.

Кроме того, можно вызывать и другие привычные команды управления файлами (например, del).

К несомненным преимуществам использования провайдера можно отнести:

естественность представления структуры директории — директория AD по своей природе иерархична и похожа на файловую систему;

удобство нахождения объектов — применять cd и dir куда удобнее, чем составлять запрос на языке LDAP.

Из недостатков бросаются в глаза:

* сложность внесения изменений в объекты — провайдер помогает легко добраться до объекта, но чтобы что-либо поменять, нам опять приходится использовать все те же директорные объекты, что и в методе ADSI, а для этого надо оперировать на низком уровне служебных методов и атрибутов AD;
* необходимость дополнительной установки — провайдер не входит в состав PowerShell, и для его применения необходимо скачать и установить расширения PowerShell;
* третьестороннее происхождение — расширения PowerShell не являются продуктом компании Microsoft. Они созданы энтузиастами проекта. Вы вольны их использовать, но за технической поддержкой придется обращаться не в Microsoft, а на сайт проекта.

## 1.4 Командлеты AD

Кроме описанного выше провайдера, для работы с AD существует и набор командлетов (часто называемых также AD cmdlets или QAD cmdlets), доступный с сайта <http://www.quest.com/activeroles_server/arms.aspx>.

Командлеты состоят из стандартных глаголов операций (get-, set-, rename-, remove-, new-, move-, connect-) и существительных объектов с префиксом QAD (-QADUser, -QADGroup, -QADComputer, -QADObject).

Например, чтобы создать новую четную запись пользователя, понадобится выполнить такую команду:

New-QADUser -ParentContainer scorpio.local/Employees -Name ‘Dmitry Sotnikov’

Преимущества данного подхода таковы:

* простота — использование командлетов скрывает от вас сложность директории, ее схемы и внутренних атрибутов. Вы работаете с объектами директории на уровне понятных названий объектов (user, group, computer), их свойств (name, password, city, department) и действий над ними (get, set, remove, move, new);
* краткость и выразительность — как мы видели, большую часть действий с помощью командлетов можно выразить в виде простых и естественных однострочных операций.

Недостатками такого подхода можно считать:

* необходимость дополнительной установки — командлеты, как и провайдер, не входят в состав PowerShell, и для их использования необходимо скачать и установить соответствующую библиотеку;
* третьестороннее происхождение — командлеты для работы с AD не являются продуктом компании Microsoft. Они созданы партнером Microsoft — компанией Quest Software. Вы вольны их применять, но за технической поддержкой придется обращаться не в Microsoft, а на форумы по работе с Active Directory на сайте PowerGUI.org.

На наш взгляд, данные недостатки с лихвой компенсируются простотой и естественностью в использовании, так что практические примеры будут приведены с применением именно этого подхода.

## 1.5 Управление Active Directory

Давайте посмотрим, как PowerShell позволяет выполнять основные операции по работе с директорией AD:

* получение информации;
* изменение свойств;
* работа с группами;
* создание новых объектов;
* изменение структуры директории

Получение информации

Получение информации осуществляется в PowerShell с помощью командлетов с глаголом Get.

Например, чтобы получить список всех пользователей, наберем:

Get-QADUser

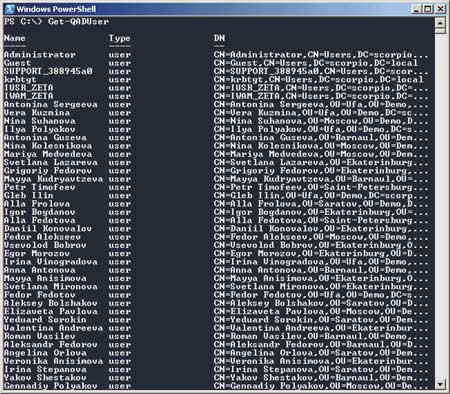
Для групп:

Get-QADGroup

Для записей компьютеров:

Get-QADComputer

Если вам нужны не все записи, а какие-то конкретные, вы можете выбрать именно их с помощью параметров команд.

**

*Получение списка пользователей*

Все группы из контейнера Users:

Get-QADGroup -SearchRoot scorpio.local/users

Все пользователи из отдела продаж московского офиса, чьи имена начинаются на букву A:

Get-QADUser -City Moscow -Department Sales -Name a\*

При этом вы можете сказать PowerShell’y, в каком виде вы хотите видеть получаемую информацию.

Таблица с именами, городами и подразделениями сотрудников:

Get-QADUser | Format-Table Name, City, Department

То же самое с сортировкой по городам:

Get-QADUser | Sort City | Format-Table DisplayName, City, Department

**

*Сортировка значений и выбор полей для вывода*

Для списочного представления той же информации просто используем команду Format-List:

Get-QADUser | Format-List Name, City, Department

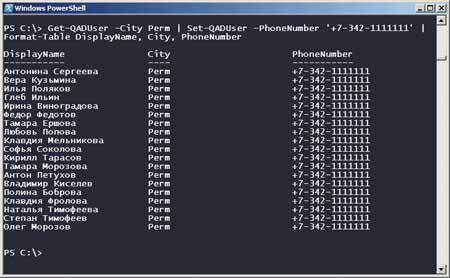
Экспортировать информацию в файл CSV (comma-separated values — значения через запятую):

Get-QADUser | Select Name, City, Department | Out-CSV users.csv

Создать отчет в формате HTML:

Get-QADUser | Select Name, City, Department | ConvertTo-HTML | Out-File users.html

Таким образом, одной строчкой простой команды PowerShell вы можете создавать сложные отчеты в удобном для вас формате.

**

*PowerShell позволяет менять атрибуты множества   
записей одной командой*

После того как мы освоились с получением информации из директории, пришла пора что-нибудь в ней поменять.

Свойствами объектов можно манипулировать с помощью команд Set-\*.

Например, поменяем мне телефон:

Set-QADUser ‘Dmitry Sotnikov’ -Phone ‘111-111-111’

Но, разумеется, куда более интересны массовые изменения. Для этого мы можем применять конвейер PowerShell, то есть получать список нужных нам объектов с помощью команд Get- и отправлять их в команду Set- для внесения изменений.

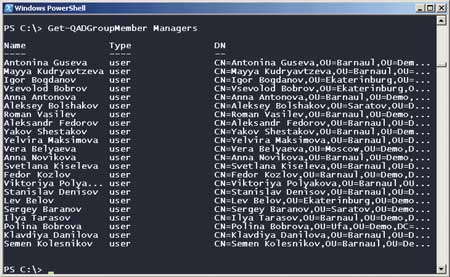
Например, наш пермский офис переехал в новое помещение. Возьмем всех пользователей Перми и присвоим им новый номер телефона:

Get-QADUser -City Perm | Set-QADUser -PhoneNumber ‘+7-342-1111111’

Для более сложных манипуляций можно использовать командлет ForEach-Object. Например, каждому пользователю присвоим описание, состоящее из его отдела и города:

Get-QADUser | ForEach-Object { Set-QADUser $\_ -Description (S\_.City + « « + $\_.Department) }

Переменная $\_ в данном примере обозначает текущий объект коллекции.

**

*PowerShell предоставляет возможности удобной работы   
с группами пользователей*

## 1.6 Работа с группами

Работа с группами и членством в них — еще одна массовая операция, которую часто хочется автоматизировать. PowerShell предоставляет такую возможность.

Получение членов группы производится с помощью командлета Get-QADGroupMember:

Get-QADGroubMember Managers

Добавить объект в группу тоже несложно:

Add-QADGroupMember Scorpio\Managers -Member dsotnikov

Аналогично удаление из группы осуществляется с помощью командлеты Remove-QADGroupMember.

Но, разумеется, наиболее полезными являются массовые манипуляции. Добавим всех менеджеров в соответствующую группу:

Get-QADUser -Title Manager | Add-QADGroupMember Scorpio\Managers

Скопируем членство в группе:

Get-QADGroupMember Scorpio\Managers | Add-QADGroupMember Scorpio\Managers\_Copy

Используем фильтр, чтобы скопировать не всех членов группы, а только тех, кто отвечает определенному критерию (например, находится в нужном регионе):

Get-QADGroupMember Scorpio\Managers | where { $\_.City -eq ‘Ekaterinburg’} | Add-QADGroupMember Scorpio\Ekaterinburg\_Managers

Обратите внимание, как мы отфильтровали пользователей с помощью команды where и логического условия (логический оператор -eq — это оператор равенства в PowerShell, *от англ.*equals).

Создание объектов

Создание объектов, как мы уже видели, осуществляется командами New:

New-QADUser -ParentContainer scorpio.local/Employees -Name ‘Dmitry Sotnikov’

New-QADGroup -ParentContainer scorpio.local/Employees -Name ‘Managers’ -Type Security -Scope Global

Вы можете установить и любые другие атрибуты в процессе создания записи:

New-QADUser -ParentContainer scorpio.local/Employees -Name ‘Dmitry Sotnikov’ -samAccountName dsotnikov -City ‘Saint-Petersburg’ -Password ‘P@ssword’

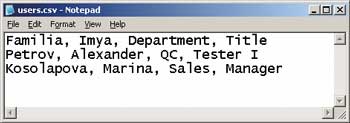
Чтобы активировать запись, просто отправьте ее по конвейеру в Enable-QADUser (не забудьте установить пароль — иначе операция не пройдет):

New-QADUser -ParentContainer scorpio.local/Employees -Name ‘Dmitry Sotnikov’ -Password ‘P@ssword’ | Enable-QADUser

Можно прочитать список пользователей и из файла. Например, если у нас есть файл, в котором через запятую перечислены атрибуты новых пользователей, то мы можем смело отправлять их на создание с помощью Import-CSV:

Import-CSV new\_users.csv | ForEach-Object { New-QADUser -ParentContainer scorpio.local/users -Name ($\_.Familia + ‘, ’ + $\_.Imya) -samAccountName ($\_.Imya[0] + $\_.Familia) -Department $\_.Department -Title $\_.Title}

Обратите внимание на то, что мы на лету составляем название учетной записи из фамилии и имени пользователя.

**

*Пример использования файла импорта   
записей*

**Изменение структуры директории**

И наконец, конечно же, можно управлять структурой директории.

Например, можно создавать новые контейнеры:

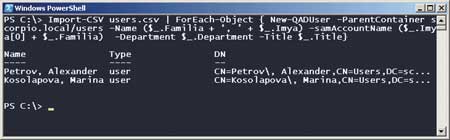
New-QADObject -type OrganizationUnit -ParentContainer scorpio.local -Name NewOU

и перемещать в них объекты по одному:

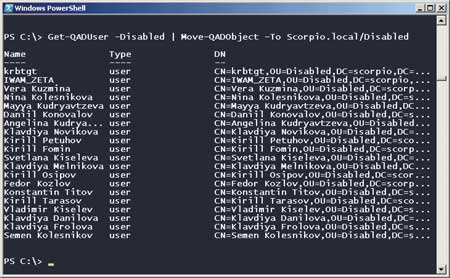
Move-QADObject MyServer -To scorpio.local/servers

или оптом:

Get-QADUser -Disabled | Move-QADObject -To scorpio.local/Disabled

**

*Импортируем файл и создаем новые учетные записи*

**

*Мы легко можем выбрать учетные записи, удовлетворяющие   
определенному критерию, и переместить их в другой контейнер*

Мы рассмотрели только малую часть сценариев по управлению активной директорией. Чтобы получить полный перечень командлетов для AD, выполните команду:

Get-Command \*-QAD\*

Чтобы получить справку по любой команде:

Get-Help Get-QADUser

Чтобы узнать, какие свойства есть у выдаваемого командой объекта:

Get-User | Get-Member

Возможности PowerShell практически безграничны, но при этом найти их достаточно легко.

Заключение

1. Аналитическая глава
2. Проектная глава

Затраты на программное обеспечение

Затраты на технические средства

Затраты на заработную плату персонала проекта, с учетом налогов на заработную плату

Расчет затрат на проект.

Приложение

#сперва нужно установить sp1 (<https://www.microsoft.com/ru-ru/download/details.aspx?id=5842>)

#на сервер, если еще не установлен.

#Далее .net 4.5 (dotNetFX45\_Full\_setup)

#После чего нужно обновить powershell до версии 5.1 (в powershell запущенном от имени администратора выполняеем Set-ExecutionPolicy bypass, и запускаем скрипт установки из архива powershell51.zip)

#при запуске скрипта poweshell длжен быть запущен от имени администратора

#при запуске без ключа, будут добаляться и обновляться пользователи

#при запуске с ключом -del будут удалться пользователи

import-module activedirectory

#переменные:

#путь до CSV файла

$pathToCSV=".\users.csv"

#пароль который будет задан новым пользователям

$defpass="As12345^"

#указываем наш домен

$domain="@matrix.ru"

#функция транслитерации

function global:Translit {

    param([string]$inString)

    $Translit = @{

    [char]'а' = "a"

    [char]'А' = "a"

    [char]'б' = "b"

    [char]'Б' = "b"

    [char]'в' = "v"

    [char]'В' = "v"

    [char]'г' = "g"

    [char]'Г' = "g"

    [char]'д' = "d"

    [char]'Д' = "d"

    [char]'е' = "e"

    [char]'Е' = "e"

    [char]'ё' = "yo"

    [char]'Ё' = "eo"

    [char]'ж' = "zh"

    [char]'Ж' = "zh"

    [char]'з' = "z"

    [char]'З' = "z"

    [char]'и' = "i"

    [char]'И' = "i"

    [char]'й' = "j"

    [char]'Й' = "j"

    [char]'к' = "k"

    [char]'К' = "k"

    [char]'л' = "l"

    [char]'Л' = "l"

    [char]'м' = "m"

    [char]'М' = "m"

    [char]'н' = "n"

    [char]'Н' = "n"

    [char]'о' = "o"

    [char]'О' = "o"

    [char]'п' = "p"

    [char]'П' = "p"

    [char]'р' = "r"

    [char]'Р' = "r"

    [char]'с' = "s"

    [char]'С' = "s"

    [char]'т' = "t"

    [char]'Т' = "t"

    [char]'у' = "u"

    [char]'У' = "u"

    [char]'ф' = "f"

    [char]'Ф' = "f"

    [char]'х' = "h"

    [char]'Х' = "h"

    [char]'ц' = "ts"

    [char]'Ц' = "ts"

    [char]'ч' = "ch"

    [char]'Ч' = "ch"

    [char]'ш' = "sh"

    [char]'Ш' = "sh"

    [char]'щ' = "sch"

    [char]'Щ' = "sch"

    [char]'ъ' = ""

    [char]'Ъ' = ""

    [char]'ы' = "y"

    [char]'Ы' = "y"

    [char]'ь' = ""

    [char]'Ь' = ""

    [char]'э' = "e"

    [char]'Э' = "e"

    [char]'ю' = "yu"

    [char]'Ю' = "yu"

    [char]'я' = "ya"

    [char]'Я' = "ya"

    }

    $outCHR=""

    foreach ($CHR in $inCHR = $inString.ToCharArray())

        {

        if ($Translit[$CHR] -cne $Null )

            {$outCHR += $Translit[$CHR]}

        else

            {$outCHR += $CHR}

        }

    Write-Output $outCHR}

#импортируем csv файл в переменную

$csv=import-Csv $pathToCSV -Encoding OEM -Delimiter ';'

#разбираем переменную

foreach ($user in $csv)

    {

        #заносим в переменные значения из csv файла

        $fio="$($user.ФИО)"

        $surname=$fio.split(' ')[0]

        $name=$fio.split(' ')[1]

        $sname=$fio.split(' ')[2]

        $dolzhnost="$($user.должность)"

        $depart="$($user.отдел)"

        $room="$($user.'номер комнаты')"

        $phone="$($user.'номер телефона')"

        $mail="$($user.'электронная почта')"

        $id=$($user.'идентификатор')

        #переводим в транслит имя и фамилию

        $transName=Translit($name)

        $transSurname=Translit($surname)

        #отчищаем первые буквы имени

        $shortName=""

        #добаляем буквы к переменной shortname (переменная для создания логина)

        for ($i=1; $i -lt $transName.length; $i++)

        {

            #в зависимости от числа проходов цикла, добавляем i букв

            $shortName=$transName.substring(0,$i)

            #добавляем буквы имени к фамилии

            $userName=$shortName+$transSurname

            try

            {

                #проеряем, есть ли пользователь

                $user=Get-ADUser "$userName"

            }

            catch

            {

                $user=$false

            }

            #если пользователь существует

            if ($user)

            {

                #получаем id из AD

                $IDinAD=Get-ADUser $userName -Properties comment | select comment | ft -HideTableHeaders | out-string

                #если номер страхового из AD совпал с номером из csv

                if ($IDinAD -match $id)

                {

                    #если запутили скрипт без аргументов

                    if ($args[0] -eq "" -or !$args[0] )

                    {

                        #обновляем данные пользователя

                        Set-ADUser -Identity "$userName" -Surname "$surname" -DisplayName "$surname $name $sname" `

                        -OfficePhone "$phone" -EmailAddress "$mail" -Department "$depart" -Title "$dolzhnost" `

                        -UserPrincipalName "$userName$domain" -GivenName "$name" -Office "$room" -enabled $true -SamAccountName "$userName"

                        #прерываем цикл

                        break

                    }

                    #если запустили скрипт с аргументом -del

                    if ($args[0] -eq "-del")

                    {

                        #удаляем пользователя

                         Remove-ADUser -Identity $userName -Confirm:$false

                    }

                }

                #если id не совпадают, и найдено имя пользователя, идем к следующему шагу цикла

                else

                {

                }

            }

            #если пользователя не существует

            else

            {

                #и запустили без аргументов

                if ($args[0] -eq "" -or !$args[0])

                {

                    try

                    {

                        $users=get-aduser -Filter "\*" -Properties comment | select comment, name

                    }

                    catch

                    {

                        $users=$false

                    }

                    if ($users)

                    {

                        foreach ($user in $users)

                        {

                            #если у какого то пользователя есть id из csv, обновляем его

                            if ($user.comment -match $id)

                            {

                                $uname=$user.name.toString()

                                $distName=Get-ADObject -Filter 'name -eq $uname'

                                Set-ADUser -Identity "$uname" -Surname "$surname" -DisplayName "$surname $name $sname" `

                                -OfficePhone "$phone" -EmailAddress "$mail" -Department "$depart" -Title "$dolzhnost" `

                                -UserPrincipalName "$userName$domain" -GivenName "$name" -Office "$room" -enabled $true `

                                -SamAccountName "$userName"

                                Rename-ADObject $distName.DistinguishedName -NewName $userName

                            }

                        }

                    }

                        try

                        {

                        #добавляем пользователя и прерываем цикл

                        New-ADUser -Name "$userName" -Surname "$surname" -DisplayName "$surname $name $sname" `

                        -OfficePhone "$phone" -EmailAddress "$mail" -Department "$depart" -Title "$dolzhnost" `

                        -UserPrincipalName "$userName$domain" -GivenName "$name" -Office "$room" -OtherAttributes @{comment="$id"} `

                        -AccountPassword (ConvertTo-SecureString -AsPlainText "$defpass" -force) -enabled $true `

                        -ChangePasswordAtLogon $true -SamAccountName "$userName" -erroraction 'silentlycontinue'

                        }

                        catch

                        {

                        }

                        break

                }

            }

        }

    }