# 4417/5417
# System Administration

Lecture 5

User Management

East Tennessee State University
Department of Computing
Jack Ramsey, Lecturer

# Outline

Active Directory

Adding Software in Linux and updating it

User management and passwords - Linux

# Active Directory - Groups

East Tennessee State University
Department of Computing
Jack Ramsey, Lecturer

# Security Group Management

Group accounts with similar characteristics together

**Scope of influence (or scope)**

- Reach of a group for gaining access to resources in Active Directory

Types of groups and associated scopes:

- Local
- Domain local
- Global
- Universal

# Security Group Management

**Security groups**

Enable access to resources on a stand-alone server or in Active Directory

**Distribution groups**

Used for e-mail or telephone lists

# Implementing Local Groups

**Local security group**

Used to manage resources on a stand-alone computer that is not part of a domain and on member servers in a domain (non-DCs)

Create using the Local Users and Groups Microsoft Management Console (MMC) snap-in

# Implementing Domain Local Groups

**Domain local security group**

- Used when Active Directory is deployed

Manage resources in a domain

- Give global groups from the same and other domains access to those resources

Scope of a domain local group

- Domain in which the group exists
- Can convert a domain local group to a universal group

# Implementing Domain Local Groups

## Access control list (ACL)

List of security descriptors (privileges) that have been set up for a particular object

| Active Directory objects that can be members of a domain local group | Active Directory objects that a domain local group can join as a member |
|---|---|
| User accounts in the same domain | Access control (security) lists for objects in the same domain, such as permissions to access a folder, shared folder, or printer |
| Domain local groups in the same domain | Domain local groups in the same domain |
| Global groups in any domain in a tree or forest (as long as there are transitive or two-way trust relationships maintained) | |
| Universal groups in any domain in a tree or forest (as long as there are transitive or two-way trust relationships maintained) | |

**Table 4-1** Membership capabilities of a domain local group

# Implementing Global Groups

**Global security group**

- Contains user accounts from a single domain
- Can also be set up as a member of a domain local group in the same or another domain

Broader scope than domain local groups

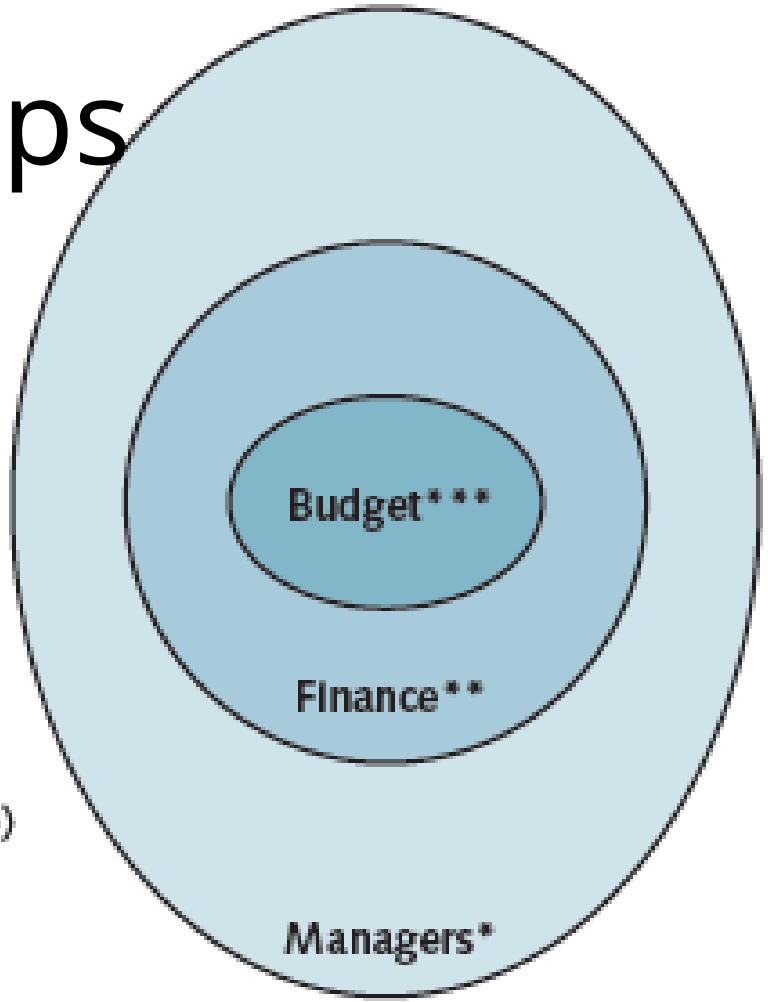
Can be nested

Typical use:

- Add accounts that need access to resources in the same or in another domain
- Make the global group in one domain a member of a domain local group in the same or another domain

# Implementing Global Groups

*Managers global group (top-level global group)
- Amber Richards
- Joe Scarpelli
- Kathy Brown
- Sam Rameriz
- **Finance global group (second-level global group)
  - Martin LeDuc
  - Sarah Humphrey
  - Heather Shultz
  - Sam Weisenberg
  - Jason Lew
  - ***Budget global group (third-level global group)
    - Michele Gomez
    - Kristin Beck
    - Chris Doyle




Nested global groups

*Courtesy Course Technology/Cengage Learning*

# Implementing Universal Groups

**Universal security groups**

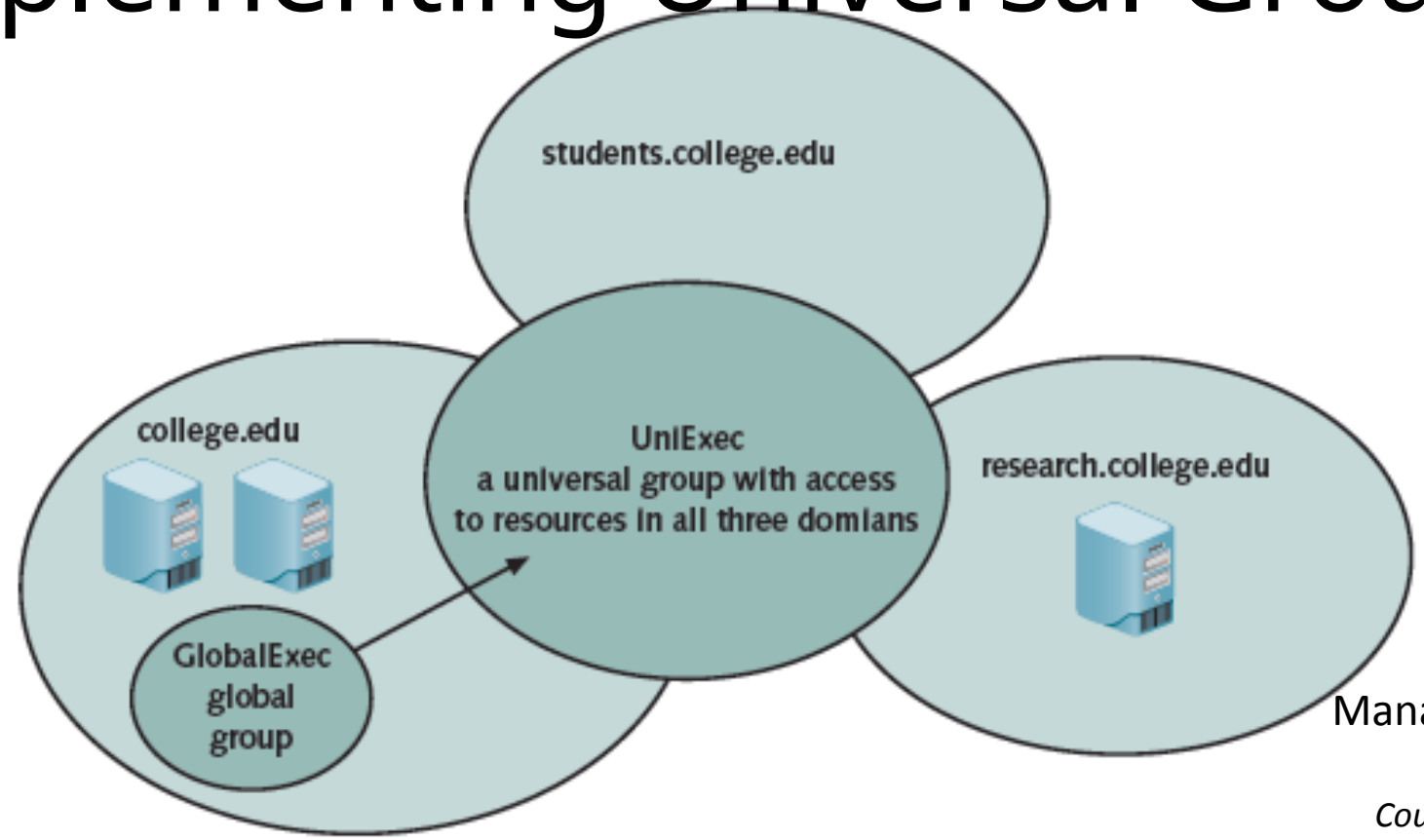
- Span domains and trees

Can include

- User accounts from any domain
- Global groups from any domain
- Other universal groups from any domain

Guidelines to help simplify how you plan to use groups

# Implementing Universal Groups




Managing security through universal and global groups

*Courtesy Course Technology/Cengage Learning*

# Properties of Groups

To edit properties:

- Double-click group in the Local Users and Groups tool for a stand-alone (non domain) or member server
- Or in the Active Directory Users and Computers tool for DC servers in a domain

Properties

- General
- Members
- Member of
- Managed by

# Planning the Delegation of Object Management

Security groups and user accounts enable an organization to delegate authority over objects

Establish and document policies

Common objects that are delegated include OUs, user accounts, and groups

# Ubuntu Basics

East Tennessee State University
Department of Computing
Jack Ramsey, Lecturer

# Installing Software `apt-get`

```
Usage: apt-get [options] command
       apt-get [options] install|remove pkg1 [pkg2 ...]
       apt-get [options] source pkg1 [pkg2 ...]

apt-get is a simple command line interface for downloading and
installing packages. The most frequently used commands are update
and install.

Commands:
   update - Retrieve new lists of packages
   upgrade - Perform an upgrade
   install - Install new packages (pkg is libc6 not libc6.deb)
```

# apt-get

Several uses

- apt-get update – updates package database
- apt-get upgrade – upgrades installed packages on the system
- apt-get dist-upgrade – attempts intelligent dependency resolution for new packages and will upgrade essential programs

# apt-get

- Several uses
  - apt-get install – install new packages. Can install multiple packages simultaneously
    - sudo apt-get install package1 package2 etc.
  - apt-get remove – removes a package, but keeps config files
  - apt-get purge – removes package and config files

# apt-cache

Used to query the package database for package information

apt-cache search – used to search for a package

```
jack@ubuntu: /proc
jack@ubuntu:/proc$ apt-cache search htop
aha - ANSI color to HTML converter
htop - interactive processes viewer
libauthen-oath-perl - Perl module for OATH One Time Passwords
jack@ubuntu:/proc$
```

# Updates for Everyone!

Updates are essential for proper administration – see CSCI 3500

Ubuntu checks every day

Also run **`sudo apt-get update`** and/or **`sudo apt-get upgrade -y`**
(the '-y' switch eliminates the 'Do you want to continue Y/n?' prompt)

# Simple User Admin

**`adduser`** vs. **`useradd`**

- **`adduser`** is a pearl script that utilizes the lower-level **`useradd`** command line tool.
- **`adduser`** more user-friendly (for the person adding the account) – prompts for password and other info
- Chooses Debian policy conformant UID and GID values, creates a home directory with skeletal configuration, running a custom script, and other features.

# What's with **sudo**?

Super User Do

**root** is locked by default

Use **sudo** to elevate privileges to execute programs

```
sudo apt-get update
```

Works the same as Window's Administrators group

# sudo

## Pros

Not logged in as **`root`**

Log entry when **`sudo`** is run

**`/var/log/auth.log`**

Easy to transfer admin rights for short term

## Cons

Command line *foo* is a little different

**`sudo ls > /root/somefile`** will not work because the shell is writing the file

# Passwords

**`/etc/passwd`** file contains all users on the system

**`'|'`** pipes the output of **`cat`** to **`grep`**

**`grep -E ":1[0-9]{3}"`** displays all 'meat users' who have been added

```
ubuntu@ubuntu: ~/scripts
ubuntu@ubuntu:~/scripts$ cat /etc/passwd | grep -E ":1[0-9]{3}:"
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
jack:x:1001:1001:Jack Ramsey,,,:/home/jack:/bin/bash
webmin:x:1002:1002:,,,:/home/webmin:/bin/bash
bob:x:1003:1003:Bob Thomas,,,:/home/bob:/bin/bash
findh:x:1004:1004:Harvey Find:/home/findh:/bin/bash
pegi:x:1005:1005:Ivan Peg:/home/pegi:/bin/bash
rindf:x:1006:1006:Florence Rind:/home/rindf:/bin/bash
mendj:x:1007:1007:Julius Mend:/home/mendj:/bin/bash
pingk:x:1008:1008:Konrad Ping:/home/pingk:/bin/bash
yorka:x:1009:1009:Alvin York:/home/yorka:/bin/bash
leonardf:x:1010:1010:Frances Leonard:/home/leonardf:/bin/bash
potterp:x:1011:1011:Peggy Potter:/home/potterp:/bin/bash
pittmana:x:1012:1012:Amy Pittman:/home/pittmana:/bin/bash
schneiderh:x:1013:1013:Homer Schneider:/home/schneiderh:/bin/bash
spencert:x:1014:1014:Travis Spencer:/home/spencert:/bin/bash
barnettr:x:1015:1015:Raymond Barnett:/home/barnettr:/bin/bash
mckenzief:x:1016:1016:Faye Mckenzie:/home/mckenzief:/bin/bash
mcdanield:x:1017:1017:Darrin Mcdaniel:/home/mcdanield:/bin/bash
baileym:x:1018:1018:Mona Bailey:/home/baileym:/bin/bash
washingtoni:x:1019:1019:Ida Washington:/home/washingtoni:/bin/bash
tatem:x:1020:1020:Maggie Tate:/home/tatem:/bin/bash
chamberss:x:1021:1021:Stacey Chambers:/home/chamberss:/bin/bash
kennedyl:x:1022:1022:Lucia Kennedy:/home/kennedyl:/bin/bash
```

# /etc/passwd

```
jack:x:1000:1000:Jack,,,:/home/Jack:/bin/bash
```

1 2 3 4 5 6 7

1. **Username**: It is used when user logs in. It should be between 1 and 32 characters in length
2. **Password**: An x character indicates that encrypted password is stored in /etc/shadow file
3. **User ID (UID)**: Each user must be assigned a user ID (UID). UID 0 (zero) is reserved for root and UIDs 1-99 are reserved for other predefined accounts. Further UID 100-999 are reserved by system for administrative and system accounts/groups
4. **Group ID (GID)**: The primary group ID (stored in /etc/group file)

# /etc/passwd

```
jack:x:1000:1000:Jack,,,:/home/jack:/bin/bash
```

1 2 3 4 5 6 7

5. **User ID Info**: The comment field. It allow you to add extra information about the users such as user's full name, phone number etc. This field use by **`finger`** command
6. **Home directory**: The absolute path to the directory the user will be in when they log in. If this directory does not exists then users directory becomes /
7. **Command/shell**: The absolute path of a command or shell (/bin/bash). Typically, this is a shell. Please note that it does not have to be a shell

# /etc/shadow

- Problem - /etc/passwd is not encrypted – BAD
- If userID is found in /etc/passwd then /etc/shadow is checked for password

jack@arwin: ~

```
pollinate:x:105:1::/var/cache/pollinate:/bin/false
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
jack:x:1001:1001:Jack,,,:/home/jack:/bin/bash
mysql:x:106:111:MySQL Server,,,:/nonexistent:/bin/false
usbmux:x:107:46:usbmux daemon,,,:/home/usbmux:/bin/false
avahi:x:108:114:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/fal
se
lightdm:x:109:116:Light Display Manager:/var/lib/lightdm:/bin/false
dnsmasq:x:110:65534:dnsmasq,,,:/var/lib/misc:/bin/false
avahi-autoipd:x:111:119:Avahi autoip daemon,,,:/var/lib/avahi-autoi
pd:/bin/false
postfix:x:112:121::/var/spool/postfix:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord
:/bin/false
```

# /etc/shadow

- A look at how the passwords are stored in /etc/shadow

jack@arwin: ~

```
jack:$6$bJzXhUoE$/xgiAPtKPYoT.7d8twdiT5t8iIHN2eJ69kJJdZ6vwMDVHORYgy
:0:99999:7:::
mysql:!:16271:0:99999:7:::
usbmux:*:16271:0:99999:7:::
avahi:*:16271:0:99999:7:::
lightdm:*:16271:0:99999:7:::
dnsmasq:*:16271:0:99999:7:::
avahi-autoipd:*:16271:0:99999:7:::
postfix:*:16271:0:99999:7:::
colord:*:16271:0:99999:7:::
dovecot:*:16271:0:99999:7:::
dovenull:*:16271:0:99999:7:::
kernoops:*:16271:0:99999:7:::
pulse:*:16271:0:99999:7:::
```

# Home Directories

Similar to ‘Documents’ folder in Windows

Usually located under **`/home/$username`**

This location does not need to be on the same partition as the OS – can even be on the network

```
cd ~
```

```
cd $HOME
```

# Start up files for Entire System

Usually end in **`.rc`**

**`profile`** is used to env variables

- /etc/profile

**`bash.bashrc`** (different for other distros)

- Called from **`profile`**
- Configure interactive variables
- Classpath, etc.
- **`/etc/bash.bashrc`**

# User Specific Files

cd $HOME

Same as **`cd /home/`*`your_user_name`*** & ***`cd ~`***

**`ls -al`**

Notice the **`.bashrc`** file

This contains user specific settings

What is the **`.bash_history`** file for? Let's see…

# Simple User Admin

Suppose we want to add a list of users at one time

We could use **`adduser`** over and over again, but that would quickly become tedious.

We can create a Bash script that will take as input a text file that has the user information for the new users

(You may want to review Chapter 2 of the Unix and Linux book -- the first part)

# Simple User Admin - Input File Format

```
adamsd Doug Adams
herbertf Frank Herbert
heinleinr Robert Heinlein
mccaffery Anne McCaffery
draked David Drake
reichsk Kathy Reichs
anthonyp Piers Anthony
```

# `adduser` - syntax

**`adduser userid -p pass -c "Fname Lname" -s /bin/bash -m`**

**`userid`** - the user's system id

**`-p pass`** - the user's initial password

**`-c "Fname Lname"`** - adds the user's name to /etc/passwd (-c = comments)

**`-s /bin/bash`** - sets the user's default shell

**`-m`** - creates a home directory for the user in /home

# museradd.sh - pseudo-code

1. Generate a default password
2. Check for the correct number of command-line arguments (2 -- “sudo museradd.sh users.txt”)
3. Check and ensure that the person running the script is using elevated permissions (“sudo”)
    1. If so, proceed
    2. If not, exit with error message
4. While still users in the input file,
    1. Read in first user information
    2. Verify that user doesn’t already exist
        1. If he/she does, display error message and proceed to the next entry
        2. If not, add the user to the system, setting the password to default, adding the user’s rw name, and the user’s home directory
        3. Force the user to change password on first login
        4. Check for success
5. Exit

# useradd.sh - code

1. Generate a default password

```
password="pass“    # puts “pass” into variable named password

# Encrypt “pass” and place the result into variable pass
pass=$(perl -e 'print crypt($ARGV[0], "password")' $password)
```

# museradd.sh - code

2. Check for the correct number of command-line arguments (2 -- “sudo museradd.sh users.txt”)

```
if [ "$1" = 1 ]; then  # check to see if there’s a second
                # argument, if not:
   echo "Usage: museradd filename"
   echo "You need to specify a valid filename!"
   exit 1
fi
```

# useradd.sh - code

3. Check and ensure that the person running the script is using elevated permissions (“sudo”)

```
if [ $(id -u) -eq 0 ]; then
```

# useradd.sh - code

4.1 While still users in the input file:

```
while read f1 f2 f3       # read the first line into $f1, $f2,
              # and $f3
do   # start the loop
```

# useradd.sh - code

## 4.2 Verify that user doesn't already exist

```
egrep "^$f1" /etc/passwd >/dev/null     # check if username
                          # already exists
if [ $? -eq 0 ]; then  # if so,
echo "User $f1 already exists! Cannot add..."
```

Note: the first line checks /etc/passwd for the presence of the userid at the beginning of a line ("^"). If there is, it will exit with a code of 0, if not, 1. The '$?' global variable stores the exit code of the last command run

# useradd.sh - code

4.2.2 If not, add the user to the system, setting the password to default, adding the user's rw name, and the user's home directory

```
else    # if the userid doesn't exist
useradd $f1 -p $pass -c "$f2 $f3" -s /bin/bash -m
```

Note: -p sets the initial password; -c adds the user's RW name to the comment field; -s sets the default shell for the user; -m creates user's home directory

# museradd.sh - code

## 4.2.3 Force the user to change password on first login

```
chage -d 0 $f1  # forces user to change password on 1st login
```

parkerm@ubuntu: /home/ubuntu/scripts

```
ubuntu@ubuntu:~/scripts$ su parkerm
Password:
You are required to change your password immediately (root enforced)
Changing password for parkerm.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
parkerm@ubuntu:/home/ubuntu/scripts$
```

# useradd.sh - code

4.2.4 Check for success

```
if [ $? -eq 0 ]; then
   echo "User has been added to system!"
else
   echo "Error! User could not be added"
fi
```

# useradd.sh - code

4. Complete the loop

```
done < $1    # when the script reaches the end of the input
        # file, the read command will return an error
        # code of 1
```

# useradd.sh - code

Tidy up - a little output signifying the script has completed.

```
echo
echo "Press Enter to continue...“
read in   # pause output until the Enter key is pressed
clear        # clear the display
```

# useradd.sh - code

Handle error if the user executing the script isn’t root

```
else    # if current user isn't root
   echo "Only root can add a user to the system"
   exit 2
fi      # end the first if statement
```

# User Management Commands

To add a user to another group:

```
usermod -G groupname userid
```

e.g.,

```
usermod -G dev alice
```

would make ‘alice’ a member of the dev group

# User Management Commands

By default, new users do not have sudo privileges

We can elevate their privileges two ways:

**`visudo`**

Edits **`/etc/sudoers file`**

```
jack@rapier: ~/Desktop
  GNU nano 2.2.6          File: /etc/sudoers.tmp


# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
jack    ALL=(ALL:ALL) ALL
```

# User Management Commands

jack = userid

ALL(1) = Rule applies to all hosts

ALL(2) = User jack can run commands as all users

ALL(3) = User jack can run commands as all groups

ALL(4) = User jack can run all commands (after providing password)

```
jack@rapier: ~/Desktop
  GNU nano 2.2.6          File: /etc/sudoers.tmp


# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
jack    ALL=(ALL:ALL) ALL
```

# User Management Commands

Easier way:

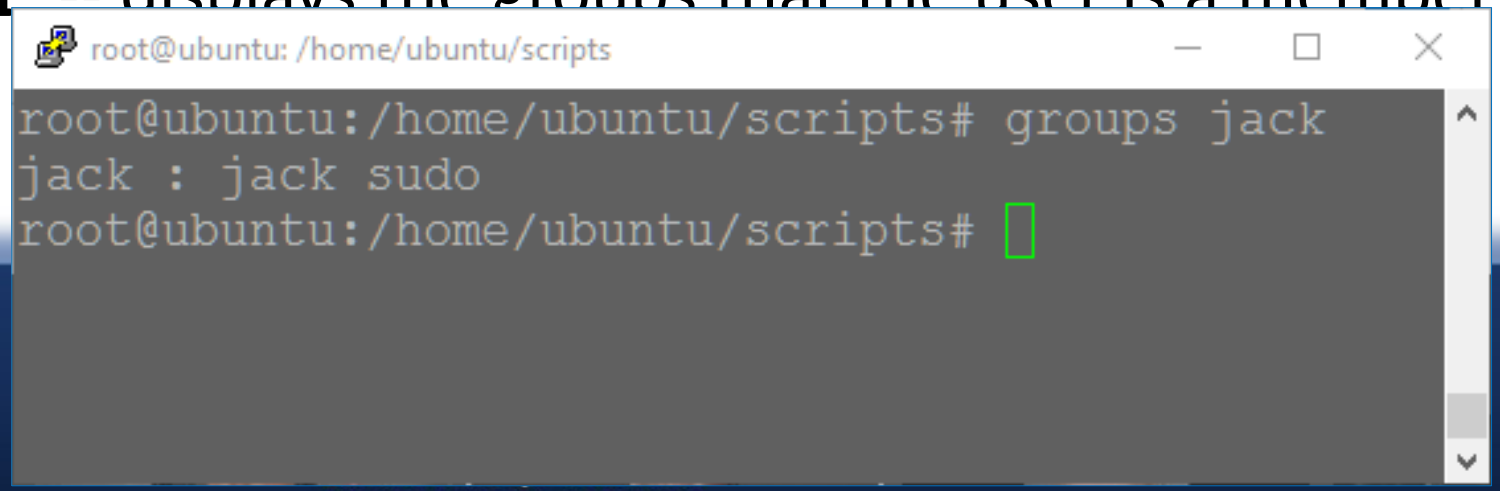
```
sudo usermod -G sudo userid
```

# User Management Commands

**`usermod --home /newdir`** -- changes the user's home directory to a new one

**`usermod --expire-date 2015-09-18 userid`** -- sets the date that the user's account will expire

**`groups userid`** -- displays the groups that the user is a member of



# User Management Commands

What if you want to run at elevated privileges for more than a single command?

```
sudo su
```

Logs in to the root user account

Can also be used to log into other user accounts

Not best practice to routinely run as root

# References

https://help.ubuntu.com/10.04/serverguide/C/user-management.html