USER AND GROUP ADMINISTRATION - UIDs, GIDs, Permissions

Commands (see shadow-utils/ pwdutils packages):
useradd, userdel, usermod
groupadd, groupdel, groupmod, gpasswd, newgrp
chmod, chgrp, chown

useradd -m username -p mypassword   ---- m option makes/provisions a home directory
useradd -D shows defaults for a new user and where /etc/skel is.
With flags can change defaults: -g for group, -s shell, -b for /home (or another directory to put user folders into).
    useradd -c Sales_Department -e 2016-12-03 -s /bin/tcsh username
     - Sets GECOS desc, expiration date of account, shell, and username

usermod -p password username --OR-- passwd alice
usermod -L username --OR-- passwd -l userName --- locks login (suspend acct so they cant log in)
usermod -U username --OR-- passwd -u userName   --- unlocks login
usermod -f {number of days} userName
    - days after a password expires until the account is permanently disabled
usermod -u {uniqueUserID} userID
usermod -d {userDirectory} userName
    New login directory - add the -m option to also move old contents
usermod -e {yyyy-mm-dd} userName
    Expire date - when account will be disabled
usermod -l {newUserName} userName
    Changes the username- <u>nothing else is changed</u>- home directory should be changed manually to match
usermod -c “comment” userName
    Or use chfn {username} to modify the details... full name, office address, phone
usermod -g -
     - Just change the user's primary group

<u>Adding a user to a group - a right way and a disastrous way</u>
The WRONG WAY - usermod -G presentations user1
  - This will add the user to the presentations group and clobber the existing group list for the user!
Use usermod -a -G groupname username to append to the user's existing grouplist
    usermod -a username -G groupname,groupname2
To verify that user1 is now a member of the presentations group, enter "id user1", "groups username", or "cat /etc/groups"
usermod -aG newgroup username is the desired option, merely adding a group to the list.
You can also add a username to their line in group manually
Group membership is not necessarily updated for current shell. A new shell is initialized with updated info

Another way to nondestructively add a used to a group is gpasswd:
gpasswd -a franklin audio ---make franklin part of audio group
gpasswd -d franklin audio ---remove franklin from audio group
cat /etc/groups ---- lists groups on the server from /etc/group

userdel and groupdel to remove a user or group
userdel -r username ---will remove user's home dir as well
userdel will not allow you to remove an account if there are running processes belonging to the account
newgrp groupname ----- to be typed by user- necessary to fully add to group for folder1 access. Also makes files created by the user to be owned by the group.

----

groupadd -g groupNumber groupName
    groupadd - g 700 print_users

groupadd -g 544 IT_Support
groupadd <groupName>
groupmod -n newName oldName
groupmod -g newGIDnum
groupdel <groupName>

[username@server ~]$ passwd ---- to change password for currently logged in user
[root@server ~]$ passwd username

It has been advised to create encrypted password first when making users:
openssl passwd -crypt
Prompts for desired password an spits out encrypted. Copy it to paste into:
useradd -c "Full Name" -m fname -p "encryptedpassword" -s "/bin/bash"

groupadd groupname ----adds new group to group listl

<u>Getting info</u>
id - show uid/ gid info. Current user if no specified username
finger/ pinkie: login name, real name, terminal name, write status, login and idle time, GECOS (office, phone)
groups ----lists groups on the server from /etc/group file
getent group support -- prints line in from /etc/group matching group "support"
getent searches ahosts, ahostsv4, ahostsv6, aliases, ethers (Ethernet addresses), group, gshadow, hosts, netgroup, networks, passwd, protocols, rpc, services, and shadow
tail /etc/passwd

userdel and groupdel to remove users or groups

passwd (the command):
-k keep non-expired auth tokens
-l lock the specified account - renders encrypted password into an invalid string (by prefixing the string with an !).
--stdin read the new password from standard input, which can be a pipe.
-u unlock the account password by removing the ! prefix. By default passwd will refuse to create a passwordless account (it will not unlock an account that has only "!" as a password). The force option -f will override this protection.
-d delete a password for an account. It will set the named account passwordless
-e expire a password for an account. Force change of password during the next login attempt.
-n minimum password lifetime, in days
-x maximum password lifetime, in days
-w number of days the user will begin receiving warnings that her password will expire
-i days which will pass before an expired password for this account will be taken to mean that the account is inactive and should be disabled
-S This will output a short information about the status of the password for a given account.

Chage- Change user password expiry information
/usr/bin/chage --- chage username --- no options is interactive mode

chage -l username ---lists these parameters
chage -m 3 username ----change min days to 3
chage -m 3 -M 60 -W 7 username
    - m minimum age in days, -M maximum age in days, -W number of warning days

for $i in `awk -F : {' print $1 '} passwd` ; do echo USERNAME: $i; chage -l $i; done

Trick found: you can set the shell of processes you want to prevent people from su'ing into as to /dev/null and put /dev/null into /etc/shells. The easier way however is to set those shells to /sbin/nologin. This can be done with "chsh -s /sbin/nologin username"

Files:
In /etc/ - passwd, group, shadow, gshadow
/etc/passwd for users
/etc/shadow for password parameters
/etc/group for groups
/etc/login.defs - holds defaults for newly created users, password expiry, umask, passwd. shadow
 - holds userid and groupid # ranges. This is in the adduser.conf file in BSD. Override with useradd -K options.
/etc/default/useradd --more default information, such as where the /etc/skel directory is

/etc/passwd   - default user accounts DB
passwd format: (the "x" means the password is in the shadow file)
username:shadow_reference:uid:gid:description(GECOS):$home:$shell
root : x : 0 : 0 : root : /root : /bin/bash
userid - 1-499 reserved for services, users start at 500
 !!'s in passwd field means none set yet.

/etc/shadow   - stores encrypted passwords, readable only to the root user:
root : dds9g739$: 13037 : 0 : 99999 : 7 : : :
username : passwordMD5 : other parameters like password age, expiry, etc
!! in password field means undefined. ! means user is locked (usermod -L username)

Username
Encrypted password — MD5
Date password last changed since epoch 1-1-1970
Days before password can be changed
Days before a password change is required
Days warning before password change
Days before the account is disabled
Date since the account has been disabled
A reserved field

----

redhat-config-users for GUI
For systems not using shadowed passwords, use pwconv and grpconv to switch

An options listing is presented when invoking add or mod commands with no arguments or username
<u>CHECK THIS</u>**

It has been advised to create encrypted password first when making users:
openssl passwd -crypt
Prompts for desired password an spits out encrypted. Copy it to paste into:
useradd -c "Full Name" -m fname -p "encryptedpassword" -s "/bin/bash"
c for canonical name, m to "make" user dir, p for password, s for shell
<u>CHECK THIS</u>**

Default User Accounts
root
bin
daemon
ftp
sshd
nfsnobody
apache
rpc
gnome

Special User Accounts
In special user accounts, the UID value for the users will be less than the default UID value, which is 500
NOTE- new UIDs and GID are assigned starting at 1000
Create a special user account with useradd -r {special user name}

Default Groups

root	0	root
bin	1	root, bin, and daemon
daemon	2	root, bin, and daemon
sys	3	root, bin, and adm
adm	4	root, bin, and daemon
tty	5	None
disk	6	root
lp	7	daemon and ip
mem	8	None
kmem	9	None
wheel	10	root
mail	12	mail
man	15	None
games	20	None
gopher	30	None
dip	40	None
ftp	50	None
nobody	99	None
users	100	None

User Private Group (UPG) - unique- created by default whenever a new user account is created

The /etc/group File
Grpname:GrpPass:GID:MembersList

Simple Permissions - chmod and umask

When you run "ls -al" you know you'll see something like this:

```
drwxr-xr-x    12 user1 staff    408 Mar 15 00:57 MyDirectory
-rwxr-xr-x+    1 user1 staff   2425 Mar 22 22:00 MyFile.py
```

The permissions block usually just use 10 bits with rw or x (for read, write, execute)
1st= file type [- or d --filename or directory, s for socket]; 2nd-4th= owners permissions
5th-7th= group owners permissions; 8th-10th= others (world)
A "-" instead of rwx is none. A "." at end of permissions list is SELinux, and a "+" indicator for any set ACLs
We'll cover those special permissions later. After permissions is link count. Files usually have a link count of 1; directories have a link count of 2 plus the number of nested directories.
After owner username is the relevant group of the owner; all members of this group have group permissions

Simple permissions with chmod

chmod u+x filename.txt (user, add x for execute permission)
chmod g+x filename.txt (group, add execute permission)
chmod o+x filename.txt (others, add execute)
chmod a+x filename.txt (all - user, group and others - give execute permissions)
You can also be quicker: "chmod ug+rwx o-rwx" (user and group get rwx, and for others subtract rwx)

It's standard in commands to use numbers instead for read write execute:
The read bit adds 4 to its total (in binary 100),
The write bit adds 2 to its total (in binary 010), and
The execute bit adds 1 to its total (in binary 001).

> Easy to remember: 4, 2, 1 = read, write, execute
> Example: add them up: user, group, and "other" permissions like this:
> chmod 764 filename.txt (means u= read, write, & execute; g=read & write; o=read)

The commands chmod and umask both use the numerical method. What's a user mask?
When you make a directory or file, it gets default permissions depending on if you are logged on as root or a regular username. It applies a mask, which basically masks out/ excludes some bits that offer permissions

Base permissions are before any of these masks are applied (pretend they don't exist for a second).
By default, the base permissions for nonexecutable files in Linux are rw-rw-rw-, or 0666.
By entering "umask 0022", the permissions assigned to all files, created until restart will be rw-r--r-- (0644)
By default, base permissions for directories in Linux are rwxrwxrwx or 0777.
By entering "umask 0022", the permissions assigned to all files, created until restart will be rwxr-xr-x (0755).

Run "umask xxx" to change the umask for the current terminal session.
Change in /etc/login.defs to make permanent.

Base permissions for files =666; for directories=777. Subtract umask to get the default permissions:
Default permissions for root (uses an auto-umask of 0022)

755	rwxr-xr-x	directory permission
644	rw-r--r--	file permission

Default permissions for general users (uses an auto-umask of 0002)

775	rwx-rwx-r-x	directory permission
664	rw-rw-r--	file permission

Quick reference for umask (these add up the same, so can also used as indirect reference for chmod numbers):

0	any permission may be set
1	setting of execute permission is prohibited
2	setting of write permission is prohibited
3	setting of write and execute permission is prohibited
4	setting of read permission is prohibited
5	setting of read and execute permission is prohibited
6	setting of read and write permission is prohibited
7	all permissions are prohibited from being set automatically

So chmod lets you set permission modes (change modes) on files and directories. Here are some other things you can do with it and some other simple tools

You can add these flags onto chmod:
-R   Modifies permissions recursively. (you WILL need this on directories you want to work on)
-f   force -to hide most error messages.
-v   verbose- info about every file processed.
-c   verbose about changes that are made to permissions.

chown {user} {file}	*Changes owner but not the group.*
chown {user:group} {file}	*Changes owner and the group.*
chown {user: } {file}	*Changes owner and group, with new group being the user's login group.*
chown {:group} {file}	*Changes group but not owner. Equivalent to using the chgrp command.*

The chattr Command (lsattr has some of these i.e -R lists recursively)
chattr -i <filename> ; lsattr
+/-I   Add or remove the read-only attribute from a file. No change, rename or deletion
When using system-config-users GUI, these parameters are not honored
-a - can only be appended
-A - access time can't be changed
-c - automatically compressed
-D - changes on dir are synchronously written to disk
-S - changes on file are synchronously written to disk
-e - extends for mapping blocks onto disk
-R   Recursively changes the attributes of directories and their contents.
-V   verbose
-v   Set a version number of a file.

lsattr-specific options
-a   Lists all files in directories.
-d   Lists directories like files, instead of listing their contents.

SETUID - P*ermissions* allow executing file as owner
Sometimes a program needs to be run with privileges of a specific user
Using "chmod 4760 filename" directs that it always executes as the owner
When listing with ls, an 's' will show in the user permissions listing in place of 'x'

SETGID - P*ermissions* allow executing file as group member.
Sometimes a program needs to be run with privileges of a specific group
Using "chmod 4760 filename" directs that it always executes as a group member
When listing with ls, an 's' will show in the group permissions listing in place of 'x'
Example: /usr/bin/wall. Set so the tty group can send messages.
chmod 2775 /directory

On directories, will enforce the same group permissions on items created within the directory structure
Setting the setgid permission on a directory *only* affects the group ID of new files and subdirectories created after the setgid bit is set, and is <u>not</u> applied to existing entities.
Set the setgid bit on existing subdirectories like this:
find /path/to/directory -type d -exec chmod g+s '{}' \;

Sticky Bits - restricted deletion flag - only the owner of a file can delete the file or directory
- to keep people from deleting other users files in /tmp or other folders
- chmod 1777 /sales   or chmod +t
- ls will show rwt instead of rwx or rw- in others (world) permissions

The Immutable Flag - set on filed (like configuration files) that should not be modified. A single directory can have a mix of mutable and immutable files and subdirectories. Also, an immutable subdirectory can have mutable files.
This is set with the chattr utility.

Special Permission Commands

chmod u[+/-/=] s {file}	Sets the SUID for a file. i.e. u+s
chmod g[+/-/=] s {directory}	Sets the SGID for a directory. g+s
chmod o[+/-/=] t {file}	Sets the sticky bit for a file. o+t
chattr +/-i {file or directory}	Sets the immutable flag for a file or directory. +i

Just like with rwx being 4, 2, 1, setuid=4, setgid=2, sticky=1
To set the setuid bit along with permissions 766: chmod 4766 filename
To set the setgid bit along with 776: chmod 2776 filename
To set sticky bit along with 766: chmod 1776 fileanme
When you do an "ls-al" permissions will show similar to -rwS--------

Beware: Root access and privilege escalation
Use the lowest permissions needed to accomplish a task. It is recommended not to give a file SUID or SGID as the root user. A user with fewer privileges often can be configured to perform the task, so SUID as THAT user instead.

Watch for back doors. If the user runs a program with the SUID set to root, then the user retains root as the effective UID when the user goes through the back door. Never set UID or GID on programs that enable you to shell out, or that have multiple entrances and exits.

Consider "chown root filename.txt && chmod u+s" (i.e., setuid as root)
Any user, binary, or script with access to this file could gain root access to the system (can set or escalate it's privileges)

Find by permissions (to find setuid and setgid)
find / -perm -u+s ---- files that have setuid
find / -perm -g+s ---- files that have setgid
find / -perm -4000 ---- files with sticky

<u>Creating a shared work directory:</u>
1. Create the shared directory ('/<filesystem>/<share-dir>')
2. Create the group and populate it with the users who are to write to the directory
3. Set up ownership of the specified directory
   `chown nobody:<accessgroup> /<filesystem>/<share-dir>`
4. Set permissions on <share-dir> w/sgid bit:
   `chmod 2770 /<filesystem>/<share-dir>`
5. Also set the 'sticky bit':
   `chmod 3770 /<filesystem>/<share-dir>`

Common SUID/GUID/Sticky examples
4700 - SUID on an executable file owned by "root"
A user named "tails" attempts to execute the file. The file owner is "root," and the permissions of the owner are executable -- so the file is executed as root.
Without SUID the user "tails" would not have been able to execute the file, as no permissions are allowed for group or others on the file. A default use of this can be seen with the /usr/bin/passwd binary file.

2070 - GUID on a directory named "planets" owned by the group "doctors" and the user "root"
A user named "tails" who belongs primarily to the group "tails" but secondarily to the group "doctors" makes a new file or directory named "neptune" under the directory named "planets." The group ownership of the new file or directory named "neptune" inherits "doctors."
Without GUID the group ownership would have been "tails."

1077 - sticky bit on a directory named "planets" owned by the group "doctors" and the user "root"
A user named "tails" creates a file named "neptune" under the directory named "planets." A user named "knuckles" attempts to delete the file named "neptune" but he cannot, since he is not the owner.
Without sticky bit "knuckles" could have deleted the file, because the directory named "planets" allows read and write by others. A default use of this can be seen with the /tmp folder.

3077 - sticky bit with GUID on a directory named "planets" owned by the group "doctors" and the user "root"
A user named "tails" who belongs to the group "doctors" creates a file or directory named "neptune" inside the directory named "planets." A user named "knuckles" who also belongs to the group "doctors" cannot delete, rename, or move the file or directory named "neptune," because he is not the owner. However, if "neptune" is a file, then "knuckles" can edit it.
If sticky bit and GUID had not been set, the user "knuckles" could rename, move, or delete the file named "neptune" because the directory named "planets" allows read and write by others. Sticky bit and GUID could be combined with something such as a read-only umask or an append only attribute

Access Control Lists (ACLs)
Permissions to more than one group or user on file or directory
Default permissions for newly created files and directories
ACL mount options set in either /etc/fstab or systemd
Tune2fs for extfs to put mount options in file system itself instead of a file.
Is an XFS default mount option
Setfacl to set and getfacl to show

Sample output from getfacl sampledir
```
# file: sample_dir
# owner:  joe
# group: account
# flags: -st
user:: rwx
group:: rwx
other:: - - -
```
(-st is sticky and setgid)

<u>ACL mechanism is not very flexible once it is applied, so it is strongly advised that general permissions be set as much as possible before setting up any ACLs on a directory</u>

Copy ACL from one file to another: getfacl file_one | setfacl -f file_two

Sample output of getacl:
```
1:  # file: somedir/
2:  # owner: lisa
3:  # group: staff
4:  user::rwx         #BASE ACL entry: USER
5:  user:joe:rwx              #effective:r-x    --named user entry for joe
6:  group::rwx                #effective:r-x            #BASE ACL entry: GROUP
7:  group:cool:r-x                          # named group entry for group "cool"
8:  mask:r-x                   # effective rights mask
9:  other:r-x                    #BASE ACL entry: OTHER
10:  default:user::rwx
11:  default:user:joe:rwx       #effective:r-x
12:  default:group::r-x
13:  default:mask:r-x
14:  default:other:---
```

Lines 4, 6 and 9 are called the *base ACL entries*.
Lines 5 and 7 are *named user* and *named group entries.*
Line 8 is the *effective rights mask,* which limits the effective rights granted to all groups and to named users. (The file owner and others permissions are not affected by the effective rights mask; all other entries are
Lines 10 through 14 display the default ACL associated with this directory. Directories may have a default ACL. Regular files never have a default ACL.

For each ACL you apply you need *<u>two</u>* commands: set for existing file, and another to make a default
setfacl -R –m g:sales:rx /data/account
setfacl –m d:g:sales:rx /data/account   <---d for default, needed for any new items that will be added to directory
-R (recursive) -m (modify) need to be in that order

```
# file: sample_dir
# owner:  joe
# group: account
# flags: -st
user:: rwx
group:: rwx
group: sales: r-x
```

```
mask:: rwx
other:: - - -
```

If you then create a subfolder, and a file within that folder, here are the getfacl output you will get:

```
# file: sample_dir
# owner:  joe
# group: account
# flags: -st
user:: rwx
group:: rwx        # effective rw-
group: sales: r-x        # effective r--
mask:: rw-
other:: - - -
```

Note- files in a subdirectory probably will have an "x" permission stripped, since this is a blanket default imposed by the mask (that no file is made executable automatically). Mask is not displayed in ACL tools as it is in umask command (for example). See above that the mask is displaying as rw- instead of what was shown in previous examples.

setfacl -m u:username:w /DemoDir/Demo.1.txt
 - Give this user write permission on this file

setfacl -m u:lisa:r file
 - Grant user **lisa** read access to file **file**.

setfacl -m m::rx file
 - Revoke write access from all groups and all named users (using the effective rights mask) for file **file**.

setfacl -x g:staff file
 - Remove the group entry for the group **staff** from file **file**'s ACL.

getfacl --access dir | setfacl -d -M- dir
 - Copy the access ACL into the default ACL

---------------- Exercises:

Create groups account and sales:
    groupadd sales; groupadd account
Add users:
    useradd caroline; useradd bob
vim /etc/group --manually add and remove from group file
cd /data; mkdir sales; mkdir account
chgrp sales sales,; chgrp account account   -- add group ownership to directories
chmod 770 * --set default permissions on directory

Exercise instructions sales and accounts should be able to read eachother's files, do that with the ACL:

```
setfacl -R –m g:sales:rx account
setfacl –m d:g:sales:rx account
setfacl -R –m g:account:rx sales
setfacl –m d:g:account:rx sales
```

if acls are set, permissions line in ls will add a "+" at the end, i.e., drwxrwx---+
Note: OSX- ls -el shows ACLs. @ shows up on some directories instead of "+" signifying xattr on .DS_Store and Library as "com.apple.FinderInfo"

DISK QUOTAS

for quotas, add to fstab
/dev/sdb5 /mnt/data ext4 defaults, usrquota 0 1
quotacheck -cug /mnt/data   scan disk usage and create quota management file
quotaon -a
repquota -a to check

quota format is used block soft, block hard, "", inode soft, inode hard
5120 10000 15000 0 500 700
edit with edquota -u username

Quotas
- Quotas are set on a per-filesystem basis (set # of blocks or # of inodes
- can be defined for both users and groups
- quota.rpm has to be installed

Setup:
- enable quota support in /etc/fstab/
change defaults to "defaults, usrquota, grpquota" for specified volume
- remount filesystems with quotas
mount -o remount /
- Create quota DB files and generate disk usage tables
quotacheck -mcug /   ----creates /aquota.user and /aquota.group
quotacheck -mavug /   ---- v = verbose c (above) is not always necessary
- Assign quota policy
edquota username ----- set blocks/inodes, soft limits (warning user), hard limits (cant write)
- Check the quotas
quota username
- remount volume ---------- mount -o remount /

quotaon -av to turn quotas on

Later to check quotas
quotacheck -mcug /   ----creates /aquota.user and /aquota.group
mentions using cron to run several times a day by adding quotacheck -avug in /etc/cron hourly, daily
quotacheck -avcm   ---all filesystems, verbose, create, and work while mounted
This updates the quota files to audit usage- aquota.user and aquota.group

edquota username   ------ enter, the :q to get out of vi.   At cmd line, type export EDITOR=nano for nano - then edquota username again
- display of number of blocks and inodes listed is what is currently in use by user
- hard and soft- 0 means no quota enforced
-edquota -t lets you set times for hard and soft limits
soft= warning, hard= limit. I nodes naturally means how many files

quota username returns line displayed in edquota (above) for one user only
repquota -a dumps quota report
repquota -s filesystem ---sends to screen a quota report
repquota -s / | less
-blocks are measured on 1k increments, so 20000 blocks is about 20mb

warnquota.conf --contains script to send email to users about quota usage

Questions you may be asked

- Trying to open a file as root and it doesn't work- give a permissions denied message!
  - if the user is trying to open a file without preceding it with cat or another command to run it, it may think the user wants to run a file as an executable. The executable bit is not going to be turned on, on a *.conf file!
  - are the actual permissions allowing read, write, or whater? You may be root, but you still have to modify permissions on files!
  - Extended ACLs may interfere, and SELinux restrictions will always say it is SELinux preventing actions.