User Administration

Managing passwords

Change current user password, prompt for current passwd - `passwd`
If you can sudo, run `sudo passwd <user>` to change a user password without prompt for current password, and with no security restrictions (min length, difficulty, etc)

Removing users

To remove a user, run `sudo userdel username`. To remove a user *and* their files within their `/home/username/` directory, run `sudo userdel -r username`

Adding users

> For a useful script to speed up this process when adding multiple users, skip to the end of this guide.

Run the following commands to create a new user on Linux -

These commands assume you are root, on a new host, so you do not need to prefix them with `sudo`, if you are not root you will need to run `sudo adduser <username>`, etc.

```
adduser username
Adding user `username' ...
Adding new group `username' (1000) ...
Adding new user `username' (1000) with group `username' ...
Creating home directory `/home/username' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for username
Enter the new value, or press ENTER for the default
        Full Name []: # You can leave all of this blank, or not
        Room Number []:# Your choice, really
        Work Phone []:
```

```
        Home Phone []:
        Other []:
    Is the information correct? [Y/n] y
```

Configuring Sudo

Now, we need to configure the user for sudo access, so we set our preferred text editor and use `sudo -E` to preserve our user's environment settings while running commands as sudo.

```
# Set vim as our preferred editor
export EDITOR=/bin/vim && export VISUAL=/bin/vim
# Edit the sudoers file, preserving our current user's environment settings
sudo -E visudo
```

Find the section within `/etc/sudoers` called user privilege specification

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

Modify the file by adding the user to the section as it appears below, granting all permissions -

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

It's considered better practice to override the `/etc/sudoers` file by running `sudo visudo -f /etc/sudoers.d/mySudoers` - This command will allow us to store our changes in a file independent from the default sudoer configuration, and also complies with the idea that `/etc/sudoer` is a *sequential configuration,* which means the order in which settings are applied is crucial to how they are interpreted by our system.

If you feel your sudoer settings are being ignored, consider moving their location in `/etc/sudoers` to the end of the file, or use the command above to create a separate configuration, securing the default settings in the even that a mistake is made, we will still be able to authenticate using sudo. Save `/etc/sudoers` and quit, but note that you will need to logout and login again for your changes to take effect.

If you configured sudo access for your user, make sure you follow the next section to ensure they are added to the relevant `sudo` *group*

Configure Group Access

> Looking to check current group members? `sudo groupmems -l -g groupname`

> Want to add a single user to a single group? `sudo usermod -aG groupname username` will `-a` append the user to the given group. The `-G` option alone will remove the user from all groups other than the one provided.

Run `vigr` in the terminal and add your new username created to the sudo group, and any other groups you may want. This is the same thing as modifying the configuration file `/etc/group` with your preferred editor and saving it. (Docker is a common group that users will need added to - Don't run your containers as root by running `sudo docker`)

```
...
tape:x:26:
sudo:x:27:USERNAME,USERNAME2,USER3
audio:x:29:
docker:x:30:USERNAME,USER3
...
```

When saving `/etc/group`, you'll get some output warning you about consistency between a shadow configuration file. Go ahead and edit it to mirror your changes, and ignore the final warning about the `/etc/group` file consistency since we just came from modifying that file.

```
vigr
You have modified /etc/group.
You may need to modify /etc/gshadow for consistency.
Please use the command 'vigr -s' to do so.


vigr -s
You have modified /etc/gshadow.
You may need to modify /etc/group for consistency.
Please use the command 'vigr' to do so.
```

Securing User / Group IDs

You should change your user and group IDs from the default sequential values we can assume Linux has distributed for us. To do this, choose and valid ID and edit the following commands to suit your needs -

```
# Change user and group IDs
sudo usermod -u 1234 user
sudo groupmod -g 4321 usergroup
```

```
# Make sure you edit all the old permissions to reflect the above changes
# Use the old user and group IDs here
sudo find / -group 1000 -exec chgrp -h username {} \;
sudo find / -user 1000 -exec chown -h username {} \;
```

Not sure what UID and GID to choose? See the table below and choose a value that suits your needs - probably a value within an unused range. **UID and GID do not need to be the same** - This is only the case by default when adding a user via Linux Distributions such as Ubuntu, which is the one referenced / used in this guide. Feel free to specify unique values, and research more into sharing user groups for permissions in scenarios such as granting a list of employees or developers similar access.

UID/GID	Purpose	Defined By	Listed in
0	`root` user	Linux	`/etc/passwd` + `nss-systemd`
1 ... 4	System users	Distributions	`/etc/passwd`
5	`tty` group	`systemd`	`/etc/passwd`
6 ... 999	System users	Distributions	`/etc/passwd`
1000 ... 60000	Regular users	Distributions	`/etc/passwd` + LDAP/NIS/...
60001 ... 61183	Unused		
61184 ... 65519	Dynamic service users	`systemd`	`nss-systemd`
65520 ... 65533	Unused		
65534	`nobody` user	Linux	`/etc/passwd` + `nss-systemd`
65535	16bit `(uid_t) -1`	Linux	
65536 ... 524287	Unused		
524288 ... 1879048191	Container UID ranges	`systemd`	`nss-mymachines`
1879048191 ... 2147483647	Unused		

UID/GID	Purpose	Defined By	Listed in
2147483648 ... 4294967294	HIC SVNT LEONES		
4294967295	32bit `(uid_t) -1`	Linux	

Table Source - Systemd.io

You should validate all the configuration done to secure your server - for example, this could be validated by running the following commands to check UID / GID after setting them and logging into our user.

Check UID / GID

id -u username

id -g username

If you plan to stop here, be sure to login to your new user before making further changes to your system.

```
sudo su username
# Or
sudo -iu username
```

Bash Add User Script

Using the information on this page, we can create a simple bash script to handle this process for us. If you plan to add a fair amount of users to a system, automating at least the general portion of that process might be valuable to you. See the script below to automate up to this point in these instructions. Simply save it into `addusers.sh` for example, and run `sudo chmod a+x addusers.sh` followed by `sudo ./addusers.sh username 1005` where 1005 is the userID you wish to assign to your new user. Sudo is required here if you wish to assign sudo privileges to the new user.

> Want to call this from the commandline as any other command? Assuming you have the script marked as an executable placed within your `/opt/` directory, run `echo "export PATH=$PATH:/opt/" >> ~/.bash_aliases && source ~/.bashrc` You should now be able to run the script by its current name from any directory on the system - `adduser.sh` Feel free to rename it

```
#!/bin/bash
## Author: Shaun Reed | Contact: shaunrd0@gmail.com | URL: www.shaunreed.com ##
## A custom bash script for creating new linux users.          ##
## Syntax: ./adduser.sh <username> <userID>                    ##
################################################################################
```

```
############

if [ "$#" -ne 2 ]; then
  printf "Illegal number of parameters."
  printf "\nUsage: sudo ./adduser.sh <username> <groupid>"
  printf "\n\nAvailable groupd IDs:"
  printf "\n60001......61183 ⎵Unused | 65520..............65533  Unused"
  printf "\n65536.....524287 ⎵Unused | 1879048191.....2147483647  Unused\n"
  exit
fi


sudo adduser $1 --gecos "First Last,RoomNumber,WorkPhone,HomePhone" --disabled-password --uid $2


printf "\nEnter 1 if $1 should have sudo privileges. Any other value will continue and make no changes\n"
read choice
if [ $choice -eq 1 ] ; then
printf "\nConfiguring sudo for $1...\n"
sudo usermod -G sudo $1
fi


printf "\nEnter 1 to set a password for $1, any other value will exit with no password set\n"
read choice

if [ $choice -eq 1 ] ; then
printf "\nChanging password for $1...\n"
sudo passwd $1
fi
```

> The script pasted above is not updated frequently, and only exists here so the code remains relevant to the information on this page. This script can be found at gitlab/shaunrd0/klips, but the version there may have changed slightly since writing the content on this page.

Now after creating this user and following the prompts in the script above, all you'll need to do is configure the user-specific settings you wish to apply in your case.

Creating SSH Keys

The steps in the section below are for generating a SSH key for the remote user you want to use to login to your server. After completing these steps, the next section will cover adding the public key we generate to the server's `authorize_keys` file, and logging into the box remotely.

To make things clear, I will refer to the machines we configure as `A` and `B`. The goal is to provide the necessary configurations on both `A` and `B` so that a user on `A` can use SSH to login to machine `B`. Presumably, machine `B` could be a VPS hosted by DigitalOcean or some other provider, and machine `A` could be your personal laptop that you plan to use to admin this server.

Remote User Configuration

SSH should never be authenticated using passwords alone, using public keys generated by `ssh-keygen` we can authenticate based on a key we generate and distribute manually to the remote server configuratioon files, allowing our user to login to the box. This should be done with care, as a combination of sloppy `authorized_keys` files and lost or stolen keys can lead to a compromised web server!

To generate an `ed25519` key for our new user, first we should navigate to their `~/.ssh/` directory - **on machine** `A`.

```
sudo su username
cd ~/.ssh/
ssh-keygen -t ed25519
```

If you run the last above command as sudo, it will create a key for `root@host`, not the user you are logged in as.
If you are getting privelege errors, you are not in your home directory. If the `~/.ssh` directory does not exist, create it and navigate within the new directory before running the `ssh-keygen` command.

You will be asked to answer a series of questions about the key you want to generate. The general format for filename is `user_<keytype>` so if our user is called `username` the file could be named `username_ed25519`. Once answering the questions this will create a public and private key and output them into your current directory (`/home/username/.ssh`), you should keep your private key safe and never share it with anyone. Your public key is what we give to the remote server so they can verify our identity when logging in.

Once the files are generated, ensure permissions are set approprately for `.ssh/` and `authorized_keys` file (if it exists)
`sudo chmod -R 700 ~/.ssh && sudo chmod 600 ~/.ssh/authorized_keys`

Login Server Configuration

Now, **on machine** `B`, create a new user following the steps in the sections above, or feel free to use the `adduser.sh` script to handle this in one step. Login to this user, just as we did on machine `A`, and navigate to their `~/.ssh` directory. Again, is this `~/.ssh` directory does not exist, just create it and then navigate within.

```
./adduser otherusername 2000
sudo su otherusername
```

```
cd ~/.ssh
```

Note that the name of the user on machine `B` does not need to match the name of the user on machine `A`, since we can specify a username with `ssh otherusername@0.0.0.0`.

Now that we have the user created on machine `B`, create an `/home/otherusername/.ssh/authorized_keys` text file and open it for editing. Paste in the public key we generated on machine `A` found at `/home/username/.ssh/username_ed25519.pub`. This `authorized_keys` file is what will be checked for approved keys when logging into machine `B` with a certain username. If the user requesting to login uses any key within it's `/home/otherusername/.ssh/authorized_keys` file, login access is granted.

Once the files are generated, ensure permissions are set approprately for `.ssh/` and `authorized_keys` file
```
sudo chmod -R 700 ~/.ssh && sudo chmod 600 ~/.ssh/authorized_keys
```

Using Putty with OpenSSH Keys

> This section is outdated, as I no longer use Putty for SSH on Windows. When working on Windows, I tend to run a Linux VM on a seperate monitor, and I just use the VM to ssh around to boxes I own. I just find this to be easier for me personally. As an alternative, you could probably just download and use the Ubuntu application on the microsoft store, and configure SSH as you would on Linux. This would save system resources required to run the VM, if all you need is a terminal.

At some point when a password is used in key generation, ssh-keygen generates openssh private key which doesn't use cipher supported by puttygen.

ssh-keygen doesn't provide option to specify cipher name to encrypt the resulting openssh private key.

There is a workaround: remove the passphrase from the key before importing into puttygen.

Create a copy of the key to temporarily remove the password
```
cp ~/.ssh/id_ed25519 ~/.ssh/id_ed25519-for-putty
```

import the copied key, using the -p argument to specify a request to set a new password, and -f to specify the import keyfile.

```
ssh-keygen -p -f ~/.ssh/id_ed25519-for-putty
Enter old passphrase: <your passphrase>
Enter new passphrase (empty for no passphrase): <press Enter>
Enter same passphrase again: <press Enter>
```

using some command, view the text contents of the private key generated.

```
cat id_ed25519-for-putty
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZWQyNTUxOQ
AAACCGyjniPP1oVCXqkdCeCKFp+5+5cI7L79rP5RYHJ5Y6fQAAAJh3QGp1d0BqdQAAAAtzc2gtZWQy
NTUxOQAAACCGyjniPP1oVCXqkdCeCKFp+5+5cI7L79rP5RYHJ5Y6fQAAAEBJr8PzmuEN6qNyrY07Lr
LAgZRjo9efYETKqFbS2jVTQobKOeI8/WhUJeqR0J4IoWn7n7lwjsvv2s/lFgcnljp9AAAADmthcHBl
ckBrYXB1bnR1AQIDBAUGBw==
-----END OPENSSH PRIVATE KEY-----
```

copy this output from your ssh session to the machine running Putty

On the windows machine, create a .ssh directory in the users folder who wishes to SSH into the server (C:\Users\Shaun.ssh)

navigate inside the directory, and create a text file - paste the output from your private key into this file, file->saveAs In the dropdown 'save as file type', select 'All Files', be sure to end the keyfile name with the .key extension -> username_ed25519.key click save.

Open puttygen, load convert->import keys.. select the text file we created in C:\Users\Shaun.ssh\ and set the passphrase from puttygen.

Don't forget to shred and remove ~/.ssh_id_ed25519-for-putty afterwards since it is not password protected.

The new password protected key will authorize the user based on the local password set in putty, using the remote PUBLIC key stored on the server.