Module 4 - Users and groups

By Juan Medina

jmedina@collin.edu

COLLIN COLLEGE

The superuser - root

Most operating systems have some sort of superuser, a user that has all power over the system. In Linux this is the **root** user. This user has the power to override normal privileges on the file system, and is used to manage and administer the system. To perform tasks such as installing or removing software and to manage system files and directories, users must escalate their privileges to the root user.

This unlimited power, comes with responsibility. **The root user has unlimited power to damage the system**: remove files and directories, remove user accounts, add back doors, and so on. If the root user's account is compromised, someone else would have administrative control of the system. **Administrators are encouraged to log in as a normal user and escalate privileges to root only when needed.**

Becoming Super-user!

There are three ways to take on an alternate identity - not only root:

1. Log out and log back in as the alternate user. (If you know the password)
2. Use the `su` command.
3. Use the `sudo` command.

The first method we will ignore as there is nothing to learn and is almost never used unless you know all your users passwords!

The su command – substitute user

The su command is used to start a shell as another user. The command syntax looks like this:

```
1 [jmedinar@localhost ~]$ su - user_name
```

The su command allows users to switch to a different user account. If you run su from a regular user account, you will be prompted for the password of the account to which you want to switch. When root runs su, you do not need to enter the user's password.

```
[jmedinar@localhost ~]$ su - user02
Password:
[jmedinar@localhost ~]$
```

If the "-" option is included, the resulting shell session is a login shell for the specified user. This means that the user's environment is loaded and the working directory is changed to the user's home directory. This is usually what we want. If the user is not specified, the superuser is assumed.

To start a shell for the superuser, we would do this:

```
[jmedinar@localhost ~]$ su -
```

After entering the command, we are prompted for the superuser's password.

We can exit any shell or session key combination of **Ctrl+C** or by the **exit** command

It's also possible to execute a single command without opening a new interactive shell

```
[jmedinar@localhost ~]$ su - c 'command' user02
```

The quotes are important as we don't any of the arguments of our command to be considered as an argument of the su command.

The sudo command – Execute command as another user

The `sudo` command is like `su` in many ways, but has some important additional capabilities.

The administrator can configure `sudo` to allow an ordinary user to execute commands as a different user in a very controlled way. In particular, a user may be restricted to one or more **specific commands** and no others. Another important difference is that the use of `sudo` does not require access to the superuser's password. To authenticate using sudo, the user uses his own password.

```
[jmedinar@localhost ~]$ sudo my_script
```

like in the example we saw in su with sudo we can do the same as follows:

```
[jmedinar@localhost ~]$ sudo su -
```

To see what privileges are granted by sudo, use the "-l" option to list them:

```
[jmedinar@localhost ~]$ sudo -l
User me may run the following commands on this host:
(ALL) ALL
```

If a user tries to run a command as another user, and the sudo configuration does not permit it, the command will be blocked, the attempt will be logged, and by default an email will be sent to the root user.

One additional benefit to using sudo is that all commands executed are logged by default to /var/log/secure.

In Red Hat Enterprise Linux and related distributions, all members of the wheel group can use sudo to run commands as any user, including root. The user is prompted for their own password.

Configuring Sudo

The main configuration file for sudo is `/etc/sudoers`. To avoid problems if multiple administrators try to edit it at the same time, it should only be edited with the special `visudo` command.

For example, the following line from the `/etc/sudoers` file enables sudo access for members of group wheel.

```
1 %wheel ALL=(ALL) ALL
```

In this line, `%wheel` is the user or group to whom the rule applies. A `%` specifies that this is a group, group wheel. The `ALL=(ALL)` specifies that on any host that might have this file, wheel can run any command. The final ALL specifies that wheel can run those commands as any user on the system.

By default, `/etc/sudoers` also includes the contents of any files in the `/etc/sudoers.d` directory as part of the configuration file. This allows an administrator to add sudo access for a user simply by putting an appropriate file in that directory.

To enable full sudo access for the user `user01`, you could create `/etc/sudoers.d/user01` with the following content:

```
1 user01 ALL=(ALL) ALL
```

To enable full sudo access for the group `group01`, you could create `/etc/sudoers.d/group01` with the following content:

```
1 %group01 ALL=(ALL) ALL
```

It is also possible to set up sudo to allow a user to run commands as another user without entering their password:

```
1 ansible ALL=(ALL) NOPASSWD:ALL
```

While there are obvious security risks to granting this level of access to a user or group, it is frequently used with cloud instances, virtual machines, and provisioning systems to help configure servers. The account with this access must be carefully protected and might require SSH public-key authentication in order for a user on a remote system to access it at all.

Need more info?

📽 How to use the sudo command

It is not part of this Linux course to see the details of sudo configuration but here's more information if you are interested

📽 Sudo: You're Doing it Wrong

What is a user?

A user account is used to provide security boundaries between different people and programs that can run commands.

Users have user names to identify them to human users and make them easier to work with. Internally, the system distinguishes user accounts by the unique identification number assigned to them, the user ID or UID. If a user account is used by humans, it will generally be assigned a secret password that the user will use to prove that they are the actual authorized user when logging in.

User accounts are fundamental to system security. Every process on the system runs under a particular user. Every file has a particular user as its owner. File ownership helps the system enforce access control for users of the files. The user associated with a running process determines the files and directories accessible to that process.

There are three main types of user account:

- **The superuser account** is for administration of the system.
- **The system accounts** which are used by processes that provide supporting services.
- **The regular accounts** which they use for their day-to-day work.

You can use the id command to show information about the currently logged-in user.

```
[jmedinar@localhost ~]$ id
uid=1000(jmedinar) gid=1000(jmedinar)
groups=1000(jmedinar),10(wheel),971(vboxusers),973(docker)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

To view basic information about another user, pass the username to the id command as an argument.

```
[jmedinar@localhost ~]$ id root
uid=0(root) gid=0(root) groups=0(root)
```

User control files

The /etc/passwd file

The /etc/passwd file is a list of users recognized by the system. The system consults /etc/passwd at login time to determine a user's UID and home directory, among other things. It is divided up into seven colon-separated fields.

```
user01:x:1000:1000:User One:/home/user01:/bin/bash
```

1. user ID

2. The user's password.
3. The UID number for this user.
4. The GID number for this user account's primary group (1000).
5. The GECOS Field. information: full name, office, extension, home phone
6. The home directory for this user.
7. The default shell program for this user.

User ID

Must be unique, can never contain colons or newlines because these characters are used as field separators and entry separators, are case sensitive, lowercase names are recommended. random sequences of letters do not make good login names. Avoid nicknames, even if your organization is informal. Names like DarkLord and QTPie belong in front of @hotmail.com, not in a professional environment. Even if your users have no self-respect, at least consider your site's overall credibility.

Since login names are often used as email addresses, it's useful to establish a standard way of forming them. It should be possible for users to make educated guesses about each other's login names. First names, last names, initials, or some combination of these all make reasonable naming schemes.

Consider that a user should have the same login name on every machine and a particular login name should always refer to the same person.

Password

Encrypted passwords used to live in the second field long ago, but that is no longer safe; with fast hardware, they can be cracked in minutes. All versions of Linux now hide the encrypted passwords by placing them in a separate file that is not world-readable. The passwd file contains an x in the encrypted password field. The actual encrypted passwords are stored in /etc/shadow file.

UID (user ID) number

The UID identifies the user to the system. Login names are provided for the convenience of users, but software and the file-system use UIDs internally. UIDs are usually unsigned 32-bit integers.

By definition, root has **UID 0**. Most systems also define pseudo-users such as bin and daemon to be the owners of commands or configuration files. It's customary to put such fake logins at the beginning of the /etc/passwd file and to give them low UIDs and a fake shell (/bin/false) to prevent anyone from logging in as those users. To allow plenty of room for nonhuman users you might want to add in the future, we recommend that you assign UIDs to real users starting at 500 or higher.

It's not a good idea to have multiple accounts with UID 0. This setup just creates more potential security holes and gives you multiple logins to secure.

Do not recycle UIDs, even when users leave your organization and you delete their accounts. This precaution prevents confusion if files are later restored from backups, where users may be identified by UID rather than by login name.

UIDs should be kept unique across your entire organization

UID Ranges

Specific UID numbers and ranges of numbers are used for specific purposes Linux.

- UID 0 is always assigned to the superuser account, root.
- UID 1-200 is a range of "system users" assigned statically to system processes by Linux.
- UID 201-999 is a range of "system users" used by system processes that do not own files on the file system. They are typically assigned dynamically from the available pool when the software that needs them is installed. Programs run as these "unprivileged" system users in order to limit their access to only the resources they need to function.
- UID 1000+ is the range available for assignment to regular users.

Default GID number

Like a `UID`, a group ID number is a 32-bit integer. `GID 0`is reserved for the group called root or system. As with `UIDs`, the system uses several predefined groups for its own housekeeping. Groups are used primarily to share access to files. The /etc/group file defines the groups, with the `GID` field in `/etc/passwd` providing a default `GID` at login time. The `GID` is relevant only to the creation of new files and directories. New files are normally owned by your effective group, but if you want to share files with others in a project group, you must then remember to manually change the files' group owner.

GECOS field

The GECOS field is sometimes used to record personal information about each user. It has no defined syntax. Although you can use any formatting conventions you like, example:

- Full name (often the only field used)
- Email
- Office number and building
- Office telephone extension
- Home phone number

Home directory

A user's home directory is his or her default directory at login time. If the home directory is missing at login time, the system prints a message such as "no home directory" and puts the user in / but if you set the `/etc/login.defs`file DEFAULT_HOME to no, the login is not allowed to continue.

Login shell

The login shell is normally a command interpreter such as the BASH (/bin/bash), but it can be any program.

The nologin Shell

The nologin shell acts as a replacement shell for the user accounts not intended to interactively log into the system. It is wise from the security standpoint to disable the user account from logging into the system when the user acount serves a responsibility that does not require the user to log into the system. For example, a mail server may require an account to store mail and a password for the user to authenticate with a mail client used to retrieve mail. That user does not need to log directly into the system.

The /etc/shadow file

A shadow password file is readable only by the superuser and serves to keep encrypted passwords safe from prying eyes and password cracking programs. It also includes some additional account information that wasn't provided for in the original `/etc/passwd`format. These days, shadow passwords are the default on nearly all systems.

```
user01:$6$DtLCvLLbi.3E8MvT$St0R20QXrWDNXbNs3W2f7bERIp7oTQBTYA.2SAzJ3O81ilR1c/Qr.:18524:0:99999:7:::
```

The shadow file is **not** a superset of the `passwd` file, and the `passwd` file is not generated from it. You must maintain both files or use tools such as `useradd` that maintain both files on your behalf. Like `/etc/passwd`,`/etc/shadow` contains one line for each user. Each line contains nine fields, separated by colons:

1. User name is the same as in /etc/passwd.
2. The encrypted password in MD5
3. Date of last password change
4. Minimum number of days between password changes
5. Maximum number of days between password changes
6. Number of days in advance to warn users about password expiration
7. Days after password expiration that account is disabled
8. Account expiration date
9. A reserved field that is currently always empty

Password Verification

When a user tries to log in, the system looks up the entry for the user in /etc/shadow, combines the salt for the user with the unencrypted password that was typed in, and encrypts them using the hashing algorithm specified. If the result matches the encrypted hash, the user typed in the right password. If the result does not match the encrypted hash, the user typed in the wrong password and the login attempt fails. This method allows the system to determine if the user typed in the correct password without storing that password in a form usable for logging in.

Format of an Encrypted Password

The encrypted password field stores three pieces of information: the hashing algorithm used, the salt, and the encrypted hash. Each piece of information is delimited by the $ sign.

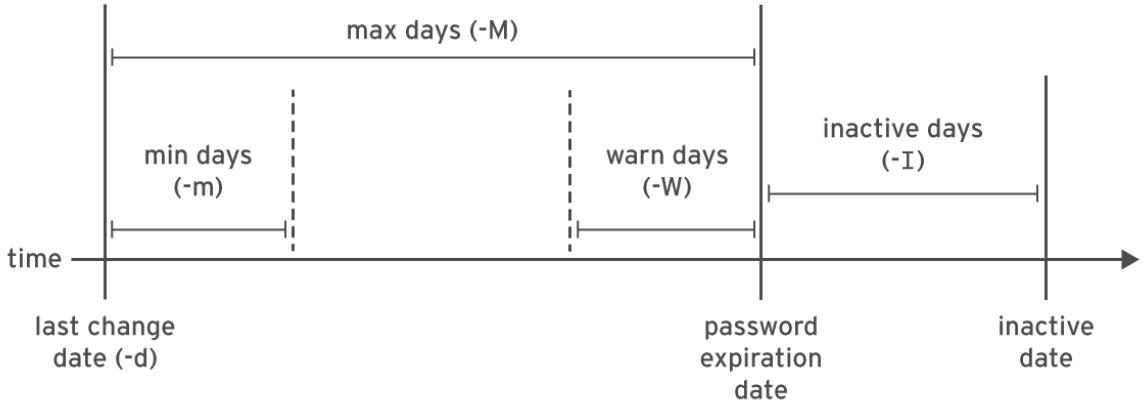
```
$6$CSsXcYG1L/4ZfHr/$2W6evvJahUfzfHpc9X.45Jc6H30E...output omitted...
```

- The hashing algorithm used for this password. The number 6 indicates it is a SHA-512 hash, which is the default in Red Hat Enterprise Linux 8. A 1 would indicate MD5, a 5 SHA-256.
- The salt used to encrypt the password. This is originally chosen at random.
- The encrypted hash of the user's password. The salt and the unencrypted password are combined and encrypted to generate the encrypted hash of the password.

The use of a salt prevents two users with the same password from having identical entries in the /etc/shadow file. For example, even if user01 and user02 both use redhat as their passwords, their encrypted passwords in /etc/shadow will be different if their salts are different.

Configuring password aging

The following diagram relates the relevant password aging parameters, which can be adjusted using the chage command to implement a password aging policy.




```
1 [jmedinar@localhost ~]$ sudo chage -m 0 -M 90 -W 7 -I 14 user03
```

The preceding chage command uses the `-m`, `-M`, `-W`, and `-I` options to set the minimum age, maximum age, warning period, and inactivity period of the user's password, respectively.

The `chage -d 0 user03` command forces the `user03` user to update its password on the next login.

The `chage -l user03` command displays the password aging details of `user03`.

The `chage -E 2019-08-05 user03` command causes the `user03` user's account to expire on `2019-08-05 (in YYYY-MM-DD format).`

Edit the password aging configuration items in the `/etc/login.defs` file to set the default password aging policies.

- PASS_MAX_DAYS sets the default maximum age of the password.
- PASS_MIN_DAYS sets the default minimum age of the password.
- PASS_WARN_AGE sets the default warning period of the password.

Any change in the default password aging policies will be effective for new users only. The existing users will continue to use the old password aging settings rather than the new ones.

The /etc/group file

The /etc/group file contains the names of Linux groups and a list of each group's members.

```
system:!:0:root,pconsole,esaadmin
```

Each line represents one group and contains four fields:

1. Group name
2. Encrypted password or a placeholder
3. GID number
4. List of members, separated by commas (be careful not to add spaces)

Same rules as passwd apply here.

Adding users

Before you create an account for a new user, **it's important that the user sign and date a copy of your local user agreement and policy statement**. Technically, the process of adding a new user consists of a few steps.

The simple process:

- Have the new user sign your policy agreement.
- Edit the passwd and shadow files to define the user's account.
- Add the user to the /etc/group file.
- Set an initial password.
- Create the user's home directory and set the proper permissions.
- Configure initial setup of the user environment
    - Copy default startup files to the user's home directory.
    - Set the user's personal information
- Add the user to your control database

Fortunately, Linux makes the whole process a breeze!

> *You must have root (administrator) authority to perform these steps.*

The useradd command

The `useradd` command implements the same basic procedure outlined above. It is configurable and we should customize it to fit our environment.

Notice:

- It takes password configuration parameters from `/etc/login.defs`, (password aging controls, encryption algorithms, mail spool files, and UID/GID ranges)
- Account defaults are set in the file `/etc/default/useradd` file and can be seen with the command useradd -D.
- All new users are placed in their own individual groups.
- The passwd file entry uses "x" as a password placeholder, the shadow file uses "!!"
- Login is disabled by default and requires a `sysadmin` to set a password for the new user.
- MD5 encryption is the default.
- A new user's home directory is populated with startup files from /etc/skel directory.

```
1 [jmedinar@localhost ~]$ useradd -c "Kriss Kross" kkross
```

Setting the password

To set or change a password, the `passwd` command is used. The command syntax looks like this:

```
1 [jmedinar@localhost ~]$ passwd kkross
```

You're prompted for the password twice. If you type a password that someone can easily guess, the `passwd` program will scold you and suggest that you use a more difficult password.

Removing users

When a user leaves your organization, that user's login account and files should be removed from the system. This procedure involves the removal of all references. If are brave enough to remove a user by hand, you may want to use the following checklist:

- Remove the user from any local user databases or phone lists.
- Remove the user from the aliases file or add a forwarding address.
- Remove the user's `crontab` file and any pending at jobs or print jobs.
- Kill any of the user's processes that are still running.
- Remove the user from the `passwd`, shadow, group, and `gshadow` files.
- Remove the user's home directory.
- Remove the user's mail spool.
- Clean up entries on shared calendars, room reservation systems, etc.
- Delete or transfer ownership of any mailing lists run by the deleted user.

Before you remove someone's home directory, be sure to relocate any files that are needed by other users. You usually can't be sure which files those might be, so it's always a good idea to make an extra backup of the user's home directory and mail spool before deleting them.

Once you have removed a user, you may want to verify that the user's old UID no longer owns files on the system using the following command to find the paths of orphaned files

```
1 [jmedinar@localhost ~]$ sudo find filesystem -xdev -nouser
```

The userdel command

The `userdel` command automates the process of removing a user. It will probably not do quite as thorough a job as you might like, unless you have religiously added functionality to it as you expanded the number of places where user-related information is stored. Linux has the `userdel.local` script but no pre- and post-execution scripts to automate things like backing up the about-to-be-removed user's files.

We can delete a user by using.

```
1 [jmedinar@localhost ~]$ userdel kkross
```

But this won't delete the user files only the account!.

To delete the account and wipe out that user's home directory as well.

```
[jmedinar@localhost ~]$ userdel -r kkross .
```

Disabling Logins

On occasion, a user's login must be temporarily disabled. A straightforward way to do this is to put a star or some other character in front of the user's encrypted password in the /etc/shadow file. This measure prevents most types of password-regulated access because the password no longer decrypts to anything sensible. But commands such as ssh that do not necessarily check the system password may continue to function, however we can use the following command.

```
[jmedinar@localhost ~]$ usermod -L <username>     # Lock Account
[jmedinar@localhost ~]$ usermod -U <username>     # Un-Lock Account
```

Modifying users

The usermod command

The usermod --help command displays the basic options that can be used to modify an account. Some common options include:

USERMOD OPTIONS:	USAGE
-c, --comment COMMENT	Add the user's real name to the comment field.
-g, --gid GROUP	Specify the primary group for the user account.
-G, --groups GROUPS	Specify a comma-separated list of supplementary groups for the user account.
-a, --append	Used with the -G option to add the supplementary groups to the user's current set of group memberships instead of replacing the set of supplementary groups with a new set.
-d, --home HOME_DIR	Specify a particular home directory for the user account.
-m, --move-home	Move the user's home directory to a new location. Must be used with the -d option.

USERMOD OPTIONS:	USAGE
-s, --shell SHELL	Specify a particular login shell for the user account.
-L, --lock	Lock the user account.
-U, --unlock	Unlock the user account.

What is a Group?

A group is a collection of users that need to share access to files and other system resources. Groups can be used to grant access to files to a set of users instead of just a single user.

Like users, groups have group names to make them easier to work with. Internally, the system distinguishes groups by the unique identification number assigned to them, the group ID or GID.

The mapping of group names to GIDs is defined in databases of group account information. By default, systems use the /etc/group file to store information about local groups.

Each line in the /etc/group file contains information about one group. Each group entry is divided into four colon-separated fields. Here is an example of a line from /etc/group

```
group01:x:10000:user01,user02,user03
```

1. Group name for this group (group01).
2. Obsolete group password field. This field should always be x.
3. The GID number for this group (10000).
4. A list of users who are members of this group as a supplementary group (user01, user02, user03). Primary (or default) and supplementary groups are discussed later in this section.

Primary Groups and Supplementary Groups

Every user has exactly one primary group. For local users, this is the group listed by GID number in the /etc/passwd file. By default, this is the group that will own new files created by the user.

Normally, when you create a new regular user, a new group with the same name as that user is created. That group is used as the primary group for the new user, and that user is the only member of this User Private Group. It turns out that this helps make management of file permissions simpler, which will be discussed later in this course.

Users may also have supplementary groups. Membership in supplementary groups is determined by the /etc/group file. Users are granted access to files based on whether any of their groups have access. It doesn't matter if the group or groups that have access are primary or supplementary for the user.

The groupadd command

If you want to create a new group, you can simply use the groupadd command.

```
[jmedinar@localhost ~]$ groupadd <group_name>
```

This will assign the next available Group ID. Then you can add users to this group with the `usermod` command.

```
[jmedinar@localhost ~]$ usermod -G mygroup myuser
```

The groupdel command

To remove a group

```
[jmedinar@localhost ~]$ groupdel my_group
```

The groupmod command

The groupmod command changes the properties of an existing group. The -n option specifies a new name for the group.

```
[jmedinar@localhost ~]$ sudo groupmod -n group0022 group02
```

- The -g option specifies a new GID.

```
[jmedinar@localhost ~]$ sudo groupmod -g 20000 group0022
```

Environment configuration

Every time we open a session the shell maintains a body of information called the environment. Data stored in the environment is used by programs to determine facts about our configuration. While most programs use configuration files to store program settings, some programs will also look for values stored in the environment to adjust their behavior.

Knowing this, we can use the environment to customize our shell experience.

What is stored in the environment?

The shell stores two basic types of data in the environment:

- Environment variables
- Shell variables.

Shell variables are bits of data placed there by bash, and environment variables are basically everything else. In addition to variables, the shell also stores some programmatic data, namely aliases and shell functions.

Examining the environment

set & printenv

To see what is stored in the environment, we can use either the **set** builtin in bash or the **printenv** program. The set command will show both the shell and environment variables, while printenv will only display the latter. Since the list of environment contents will be fairly long, it is best to pipe the output of either command into less:

```
[jmedinar@localhost ~]$ printenv | less
```

Doing so, we should get something that looks like this:

```
KDE_MULTIHEAD=false
SSH_AGENT_PID=6666
HOSTNAME=linuxbox
GPG_AGENT_INFO=/tmp/gpg-PdOt7g/S.gpg-agent:6689:1
SHELL=/bin/bash
TERM=xterm
XDG_MENU_PREFIX=kde-
HISTSIZE=1000
XDG_SESSION_COOKIE=6d7b05c65846c3eaf3101b0046bd2b00-
1208521990.996705-1177056199
GTK2_RC_FILES=/etc/gtk-2.0/gtkrc:/home/me/.gtkrc-
2.0:/home/me/.kde/share/config/gtkrc-2.0
GTK_RC_FILES=/etc/gtk/gtkrc:/home/me/.gtkrc:/home/me/.kde/share/con
fig/gtkrc
GS_LIB=/home/me/.fonts
WINDOWID=29360136
```

```
14 QTDIR=/usr/lib/qt-3.3
15 QTINC=/usr/lib/qt-3.3/include
16 KDE_FULL_SESSION=true
17 USER=me
18 LS_COLORS=no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;
   01
```

What we see is a list of environment variables and their values. For example, we see a variable called USER, which contains the value "me". The printenv command can also list the value of a specific variable:

```
1 [jmedinar@localhost ~]$ printenv USER
```

The set command, when used without options or arguments, will display both the shell and environment variables, as well as any defined shell functions. Unlike printenv, its output is courteously sorted in alphabetical order:

```
1 [jmedinar@localhost ~]$ set | less
```

It is also possible to view the contents of a variable using the echo command, like this:

```
1 [jmedinar@localhost ~]$ echo $HOME
```

The alias command

One element of the environment that neither **set** nor **printenv** displays is aliases. To see them, enter the **alias** command without arguments:

```
1 [jmedinar@localhost ~]$ alias
```

Some Interesting Variables

The environment contains quite a few variables, and though your environment may differ from the one presented here, you will likely see the following variables in your environment:

- **DISPLAY** The name of your display if you are running a graphical environment. Usually this is ":0", meaning the first display generated by the X server.
- **EDITOR** The name of the program to be used for text editing.
- **SHELL** The name of your shell program.
- **HOME** The pathname of your home directory.
- **LANG** Defines the character set and collation order of your language.
- **OLD_PWD** The previous working directory.
- **PATH** A colon-separated list of directories that are searched when you enter the name of an executable program.
- **PS1** Prompt String 1. This defines the contents of your shell prompt.
- **PWD** The current working directory.
- **TERM** The name of your terminal type.
- **USER** Your username.

How is the environment established?

When we log on to the system, the bash program starts and reads a series of configuration scripts called startup files, which define the default environment shared by all users. This is followed by more startup files in our home directory that define our personal environment. The exact sequence depends on the type of shell session is started.

A login shell session is one in which we are prompted for our username and password

Startup files for login shell sessions

- **/etc/profile** A global configuration script that applies to all users.
- **~/.bash_profile** A user's personal startup file. It can be used to extend or override settings in the global configuration script.
- **~/.bash_login** If ~/.bash_profile is not found, bash attempts to read this script.
- **~/.profile** If neither ~/.bash_profile nor ~/.bash_login is found, bash attempts to read this file.
- **~/.bashrc** file is probably the most important startup file from the ordinary user's point of view since it is almost always read.

Lines that begin with a "#" are comments and are not read by the shell. These are there for human readability. The first interesting thing occurs on the fourth line, with the following code:

```
1 if [ -f ~/.bashrc ]; then
2             . ~/.bashrc
3 fi
```

Human translation: If the file "~/.bashrc" exists, then read the "~/.bashrc" file.

We can see that this bit of code is how a login shell gets the contents of .bashrc.

The next thing in our startup file has to do with the PATH variable that is how the shell knows where to find commands when we enter them on the command line. For example, the ls command resides under /bin/ls but Linux doesn't have to look for it all over the system, instead it will check on the paths stored in the $PATH environment variable

```
1 [jmedinar@localhost ~]$ echo $PATH
```

PATH can be modified to add the directory $HOME/bin to the end of the list as follows.

```
1 [jmedinar@localhost ~]$ PATH=$PATH:$HOME/bin
```

By adding the string `$HOME/bin` to the end of the PATH variable's contents, the directory `$HOME/bin` is added to the list of directories searched when a command is entered. This means that when we want to create a directory within our home directory for storing our own private programs, the shell is ready to accommodate us.

Lastly, we have:

```
1 [jmedinar@localhost ~]$ export PATH
```

The export command tells the shell to make the contents of PATH available to child processes of this shell.

🎀 Customizing Your Terminal: Adding Color and Information to Your Prompt

🎀 Customizing Your Terminal: .bash_profile and .bashrc files

Introduction to Vim Editor

A key design principle of Linux is that information and configuration settings are commonly stored in text-based files. These files can be structured in various ways, as lists of settings, in INI-like formats, as structured XML or YAML, and so on. However, the advantage of text files is that they can be viewed and edited using any simple text editor.

Vim is an improved version of the vi editor distributed with Linux and UNIX systems. Vim is highly configurable and efficient for practiced users, including such features as split screen editing, color formatting, and highlighting for editing text.

Why you should learn vim?

Learning the Linux command line, like becoming an accomplished pianist, is not something that we pick up in an afternoon. **It takes years of practice**. The vim text editor, one of the core programs in the Linux tradition. It has a difficult user interface, but when you learn it is like seeing a master pianist sit down and begin to "play".

In this modern age with so many graphical editors so easy-to-use, why should we learn vim?

There are three good reasons:

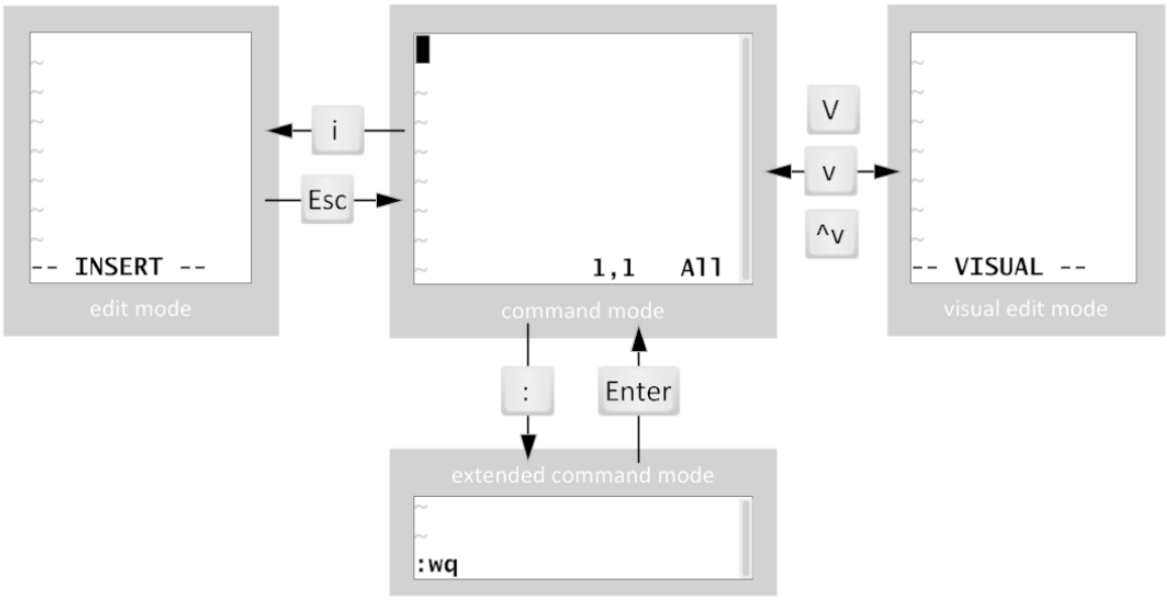
- **vim is always available.** This can be a lifesaver if we have a system with no graphical interface, such as a remote server.
- **vim is lightweight and fast.** For many tasks, it's easier to bring up vi than it is to find the graphical text editor in the menus and wait for its multiple megabytes to load. In addition, vim is designed for typing speed. As we shall see, a skilled vi user never has to lift his or her fingers from the keyboard while editing.
- **We don't want other Linux and Unix users to think we are sissies.**
  `Okay, maybe two good reasons.`

Vim Operating Modes

An unusual characteristic of Vim is that it has several modes of operation

- Command mode [`ESC`]
- Extended command mode [`:`]
- Edit mode [`i` Insert or `a` append]
- Visual mode [`v`]

Depending on the mode, you may be issuing commands, editing text, or working with blocks of text. As a new Vim user, you should always be aware of your current mode as keystrokes have different effects in different modes.



When you first open Vim, it starts in **command mode**, which is used for navigation, cut and paste, and other text manipulation.

Enter each of the other modes with single character keystrokes to access specific editing functionality:

- An `i` keystroke enters `insert mode`, where all text typed becomes file content. you can also use `a` to enter insert mode but appending text.
- Pressing `Esc` returns to `command mode`.
- A `v` keystroke enters `visual mode`, where multiple characters may be selected for text manipulation. Use `Shift+V` for multiline and `Ctrl+V` for block selection. The same keystroke used to enter visual mode (v, Shift+V or Ctrl+V) is used to exit.
- The `:` keystroke begins `extended command mode` for tasks such as writing the file on disk to save it, and quitting the Vim editor.

Entering and exiting vim

To start vim, we simply enter the following:

```
[jmedinar@localhost ~]$ vim
```

To exit, (note that the colon character is part of the command)

```
:q
```

The shell prompt should return. If for some reason, vi will not quit (usually because we made a change to a file that has not yet been saved), we can tell vi that we really mean it by adding an exclamation point to the command:

```
:q!
```

Creating new file

```
[jmedinar@localhost ~]$ vim myfile.txt
```

Inserting text

In order to add some text to our file, we must first enter insert mode. To do this, we press the **"i"** key. Afterward, we should see the following at the bottom of the screen:

```
-- INSERT --
```

Now we can enter some text. Try this:

```
Professor Medina is the coolest guy in the world! and I am a
compulsive liar!
```

> *To exit insert mode and return to command mode, press the Esc key.*

> *To exit other mode and return to command mode, press the Esc key.*

> *To exit any mode and return to command mode, press the Esc key.*

Saving Our Work

To save the change we just made to our file, we must enter a command, of course in command mode!!!. This is easily done by pressing the ":" key. After doing this, a colon character should appear at the bottom of the screen:

```
:
```

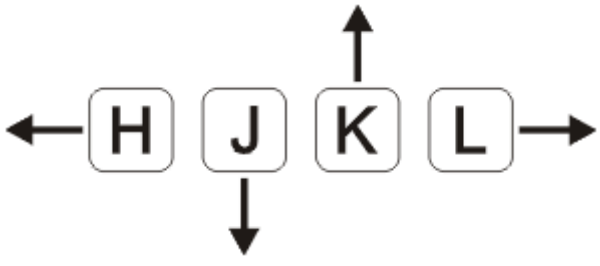
To write our modified file, we follow the colon with a "w" then Enter:

```
:w
```

and we can combine commands, for instance, to save and exit at the same time:

```
:wq
```

Cursor movement



- `w` - jump by start of words (punctuation considered words)
- `b` - jump backward by words (punctuation considered words)
- `0` - (zero) start of line
- `^` - first non-blank character of line (same as 0w)
- `$` - end of line
- Advanced (in order of what I find most useful)
    - `}` - go forward by paragraph (the next blank line)
    - `{` - go backward by paragraph (the next blank line)
    - `gg` - go to the top of the page
    - `G` - go the bottom of the page

Insert/Appending/Editing Text

- Results in Insert mode
    - `i` - start insert mode at the cursor
    - `a` - append after the cursor
    - `o` - open (append) blank line below the current line (no need to press return)
    - `O` - open blank line above the current line
- `r [char]` - replace a single character with the specified char (does not use Insert mode)
- `d` - delete with movement
- `dd` - delete the current line

Marking text (visual mode)

- `v` - starts visual mode (From here you can move around as in normal mode)
- `V` - starts line-wise visual mode
- `Ctrl+v` - start visual block mode

Visual commands

Type any of these while some text is selected to apply the action

- `y` - yank (copy) marked text
- `d` - delete marked text
- `c` - delete the marked text and go into insert mode

Cut and Paste

- `yy` - yank (copy) a line
- `p` - put (paste) the clipboard after cursor
- `x` - delete (cut) current character

Exiting

- `:w` - write (save) the file, but don't exit
- `:wq` - write (save) and quit
- `:q` - quit (fails if anything has changed)
- `:q!` - quit and throw away changes

Search/Replace

- `/pattern` - Search for pattern
- `?pattern` - search backward for pattern
- `n` - repeat the search in the same direction
- `N` - repeat the search in opposite direction
- `:%s/old/new/g` - replace all old with new throughout the file
- `:%s/old/new/gc` - replace all old with new throughout file with confirmations

A bit more of practice? Try - https://www.openvim.com/

Need more info?

🎀 Vim Tutorial

.vimrc configuration file

Here is a copy of my `.vimrc` file just make a copy in your home directory as `.vimrc` (dot included).

You can select any color schema by uncommenting the line removing the `"` character on the one you want to use.

```
"Tabulator size
   set ts=3
"File Encoding
   set fileencodings=utf-8,big5,gbk,cp936,iso-2022-jp,sjis,euc-
jp,euc-kr,utf-bom,iso8859-1
   set encoding=utf-8
   set tenc=utf-8
"Gui Font
   set guifont=DejaVu\ Sans\ Mono\ 12
   if exists("&ambiwidth")
```

```
    set ambiwidth=double
  endif
"Line numbers
  set nu
"Command History
  set history=100
  set backspace=indent,eol,start
"Check if a open file has change
  set autoread
"HighLight Searched Patterns
  set hls
"HighLight the matched string
  set incsearch
"Make vim more compatible with vi
  set nocompatible
"Indentation Level
  set autoindent  "...
"Do smart indentation
  set smartindent  "...
"set cindent
  set paste
  set noerrorbells visualbell     " be quiet
  set showmatch   "...
  set confirm
  set wildmenu
  set showcmd
  set showmode
  set ruler
  set cmdheight=2
  set gcr=a:blinkon0      "........
  set report=0    " always report changes
  set lz  " do not redraw while running macros
  set lsp=0       " space it out a little more
  set ws
  set more        "use a pager for long listings
  filetype on
  filetype plugin on
  filetype plugin indent on
  set guioptions-=T       "......
  set bsdir=buffer
  set laststatus=2
  set statusline=%f\ %y%r%1*%m%*%=%<x%02B%4vv%4cc%7l\ %P
  set ffs=unix,dos,mac
  set ff=unix
  nmap <leader>fd :se ff=dos<cr>
  syntax on
  "color codeschool
  "color default
  "color elflord
  "color morning
  "color peachpuff
  "color torte
  "color blue
  "color delek
  "color evening
```

```
"color murphy
"color ron
"color zellner
"color darkblue
"color desert
"color koehler
color pablo
"color shine
"Omni menu colors
hi Pmenu guibg=#333333
hi PmenuSel guibg=#555555 guifg=#ffffff
"set showtabline=2
map th :tabnext<CR>
map tl :tabprev<CR>
map tn :tabnew<CR>
map td :tabclose<CR>
" When using make, where should it dump the file
set makeef=error.err
set softtabstop=4       "... softtabstop . 4. .. tab .... 4 ..

set shiftwidth=4        "...... 4 ..., ..........
```