AI STUDIO

LINUX

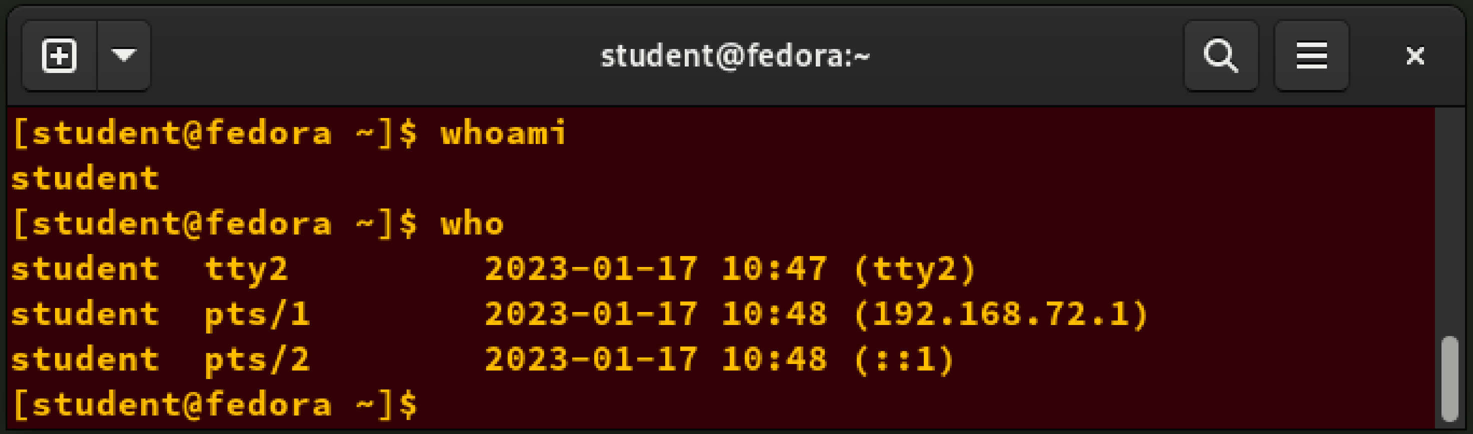
Accounts, Users and Groups

AUGUST 2025

AI STUDIO

Identifying the Current User

↘ As you know, Linux is a multi-user operating system, meaning more than one user can log on at the same time.
↘ To identify the current user, type whoami.
↘ To list the currently logged-on users, type who.
↘ Giving who the -a option will give more detailed information.



```
[student@fedora ~]$ whoami
student
[student@fedora ~]$ who
student  tty2         2023-01-17 10:47 (tty2)
student  pts/1        2023-01-17 10:48 (192.168.72.1)
student  pts/2        2023-01-17 10:48 (::1)
[student@fedora ~]$
```

AI STUDIO

User Startup Files

- ↘ In Linux, the command shell program (generally bash) uses one or more startup files to configure the user environment. Files in the /etc directory define global settings for all users, while initialization files in the user's home directory can include and/or override the global settings.
  - ↘ The startup files can do anything the user would like to do in every command shell, such as:
- ↘ Customizing the prompt
- ↘ Defining command line shortcuts and aliases
- ↘ Setting the default text editor
- ↘ Setting the path for where to find executable programs

AI STUDIO

Order of the Startup Files

↘ The standard prescription is that when you first login to Linux, **/etc/profile** is read and evaluated, after which the following files are searched (if they exist) in the listed order:

↘ **~/.bash_profile**

↘ **~/.bash_login**

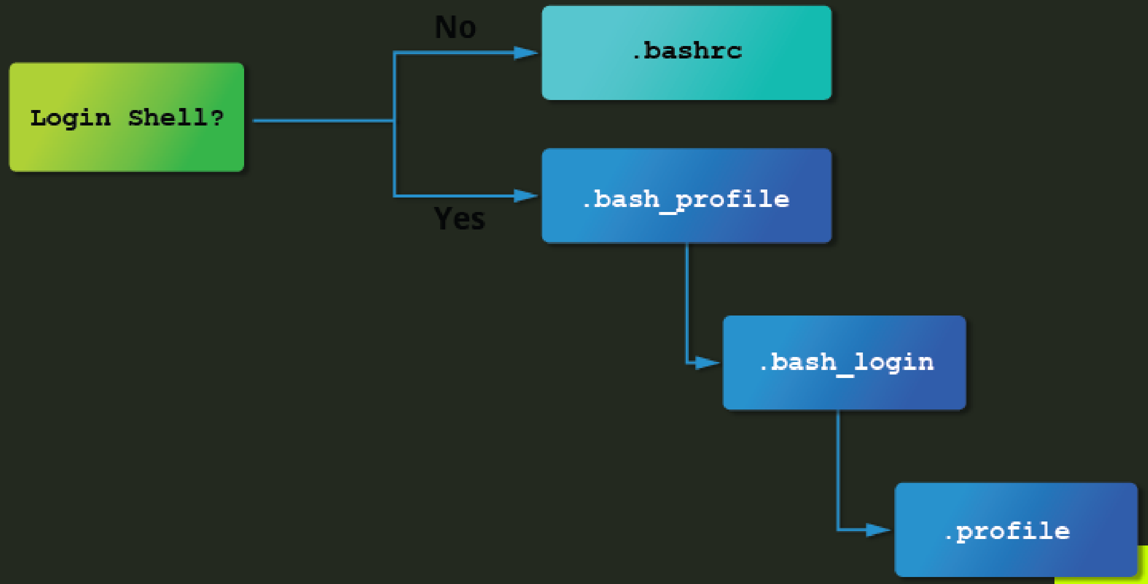
↘ **~/.profile**

↘ where ~/ denotes the user's home directory. The Linux login shell evaluates whatever startup file that it comes across first and ignores the rest. This means that if it finds **~/.bash_profile**, it ignores **~/.bash_login** and **~/.profile**. Different distributions may use different startup files.

↘ However, every time you create a new shell, or terminal window, etc., you do not perform a full system login; only a file named **~/.bashrc** file is read and evaluated. Although this file is not read and evaluated along with the login shell, most distributions and/or users include the **~/.bashrc** file from within one of the three user-owned startup files.

↘ Most commonly, users only fiddle with **~/.bashrc**, as it is invoked every time a new command line shell initiates, or another program is launched from a terminal window, while the other files are read and executed only when the user first logs onto the system.

↘ Recent distributions sometimes do not even have **.bash_profile** and/or **.bash_login**, and some just do little more than include **.bashrc**.



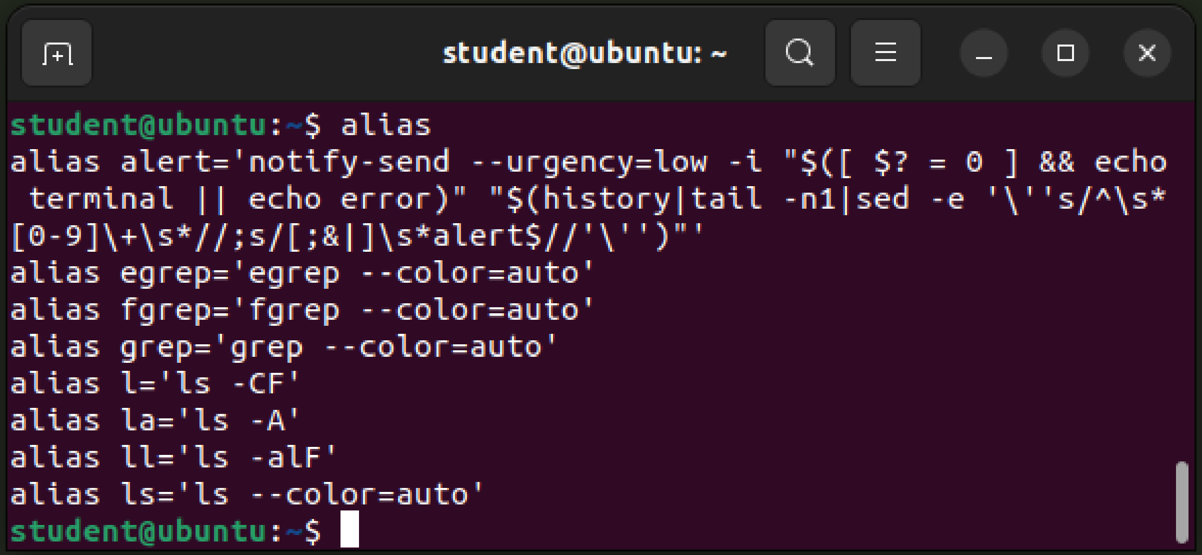
AI STUDIO

Creating Aliases

↘ You can create customized commands or modify the behavior of already existing ones by creating **aliases**. Most often, these aliases are placed in your **~/.bashrc** file so they are available to any command shells you create. **unalias** removes an alias.

↘ Typing **alias** with no arguments will list currently defined aliases.

↘ Please note there should not be any spaces on either side of the equal sign and the alias definition needs to be placed within either single or double quotes if it contains any spaces.



student@ubuntu: ~

```
student@ubuntu:~$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo
 terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*
[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
student@ubuntu:~$
```

AI STUDIO

Basics of Users and Groups

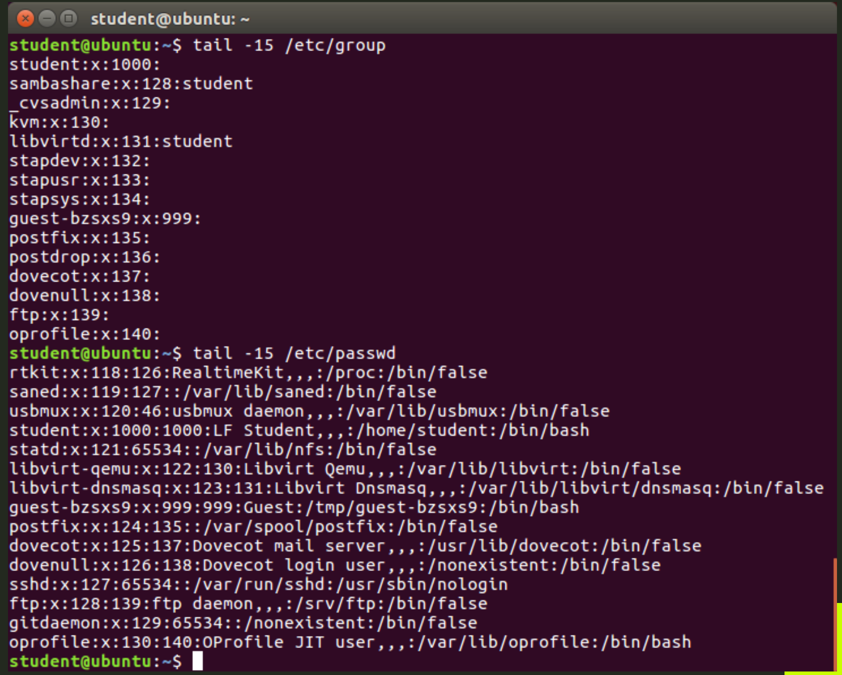
↘ All Linux users are assigned a unique user ID (**uid**), which is just an integer; normal users start with a uid of 1000 or greater.

↘ Linux uses **groups** for organizing users. Groups are collections of accounts with certain shared permissions; they are used to establish a set of users who have common interests for the purposes of access rights, privileges, and security considerations. Access rights to files (and devices) are granted on the basis of the user and the group they belong to.

↘ Control of group membership is administered through the /etc/group file, which shows the list of groups and their members. By default, every user belongs to a default (primary) group. When a user logs in, the group membership is set for their primary group, and all the members enjoy the same level of access and privilege. Permissions on various files and directories can be modified at the group level.

↘ Users also have one or more group IDs (**gid**), including a default one that is the same as the user ID. These numbers are associated with names through the files /etc/passwd and /etc/group.

↘ For example, /etc/passwd might contain john:x:1002:1002:John Garfield:/home/john:/bin/bash, and /etc/group might contain john:x:1002.


student@ubuntu: ~

```
student@ubuntu:~$ tail -15 /etc/group
student:x:1000:
sambashare:x:128:student
_cvsadmin:x:129:
kvm:x:130:
libvirtd:x:131:student
stapdev:x:132:
stapusr:x:133:
stapsys:x:134:
guest-bzsxs9:x:999:
postfix:x:135:
postdrop:x:136:
dovecot:x:137:
dovenull:x:138:
ftp:x:139:
oprofile:x:140:
student@ubuntu:~$ tail -15 /etc/passwd
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
student:x:1000:1000:LF Student,,,:/home/student:/bin/bash
statd:x:121:65534::/var/lib/nfs:/bin/false
libvirt-qemu:x:122:130:Libvirt Qemu,,,:/var/lib/libvirt:/bin/false
libvirt-dnsmasq:x:123:131:Libvirt Dnsmasq,,,:/var/lib/libvirt/dnsmasq:/bin/false
guest-bzsxs9:x:999:999:Guest:/tmp/guest-bzsxs9:/bin/bash
postfix:x:124:135::/var/spool/postfix:/bin/false
dovecot:x:125:137:Dovecot mail server,,,:/usr/lib/dovecot:/bin/false
dovenull:x:126:138:Dovecot login user,,,:/nonexistent:/bin/false
sshd:x:127:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:128:139:ftp daemon,,,:/srv/ftp:/bin/false
gitdaemon:x:129:65534::/nonexistent:/bin/false
oprofile:x:130:140:OProfile JIT user,,,:/var/lib/oprofile:/bin/bash
student@ubuntu:~$
```

AI STUDIO

Adding and Removing Users

↘ Distributions have straightforward graphical interfaces for creating and removing users and groups and manipulating group membership. However, it is often useful to do it from the command line or from within shell scripts. Only the root user can add and remove users and groups.

↘ Adding a new user is done with **useradd** and removing an existing user is done with **userdel**. In the simplest form, an account for the new user **bjmoose** would be done with:

↘ `$ sudo useradd bjmoose`

↘ which, by default, sets the home directory to /home/bjmoose, populates it with some basic files (copied from /etc/skel) and adds a line to /etc/passwd such as:

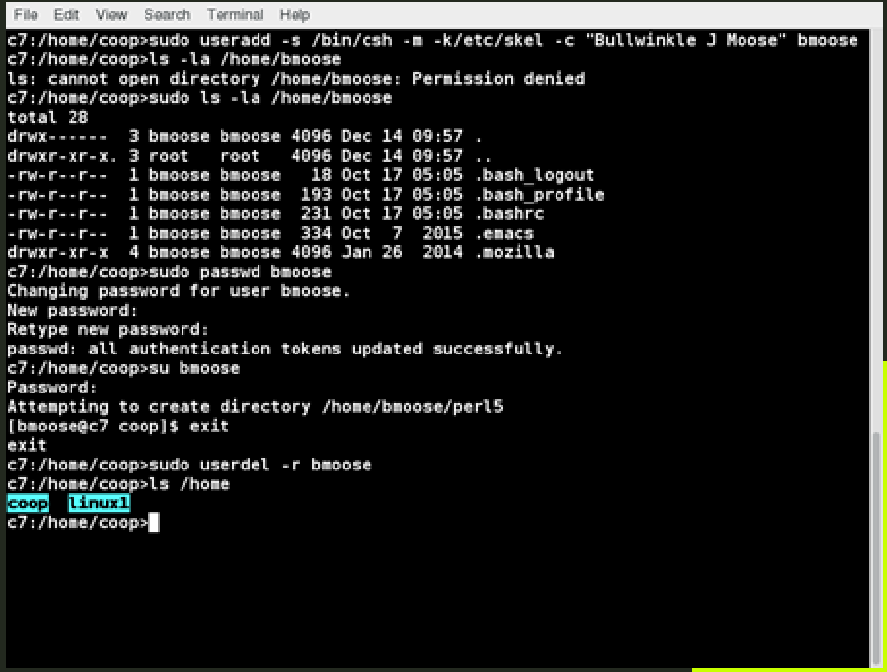
↘ **bjmoose:x:1002:1002::/home/bjmoose:/bin/bash**

↘ and sets the default shell to /bin/bash. Removing a user account is as easy as typing **userdel bjmoose**. However, this will leave the /home/bjmoose directory intact. This might be useful if it is a temporary inactivation. To remove the home directory while removing the account one needs to use the **-r** option to **userdel**.

↘ Typing **id** with no argument gives information about the current user, as in:

↘
```
$ id
uid=1002(bjmoose) gid=1002(bjmoose) groups=106(fuse),1002(bjmoose)
```

↘ If given the name of another user as an argument, **id** will report information about that other user.



```
c7:/home/coop>sudo useradd -s /bin/csh -m -k/etc/skel -c "Bullwinkle J Moose" bmoose
c7:/home/coop>ls -la /home/bmoose
ls: cannot open directory /home/bmoose: Permission denied
c7:/home/coop>sudo ls -la /home/bmoose
total 28
drwx------  3 bmoose bmoose 4096 Dec 14 09:57 .
drwxr-xr-x. 3 root   root   4096 Dec 14 09:57 ..
-rw-r--r--  1 bmoose bmoose   18 Oct 17 05:05 .bash_logout
-rw-r--r--  1 bmoose bmoose  193 Oct 17 05:05 .bash_profile
-rw-r--r--  1 bmoose bmoose  231 Oct 17 05:05 .bashrc
-rw-r--r--  1 bmoose bmoose  334 Oct  7  2015 .emacs
drwxr-xr-x  4 bmoose bmoose 4096 Jan 26  2014 .mozilla
c7:/home/coop>sudo passwd bmoose
Changing password for user bmoose.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
c7:/home/coop>su bmoose
Password:
Attempting to create directory /home/bmoose/perl5
[bmoose@c7 coop]$ exit
exit
c7:/home/coop>sudo userdel -r bmoose
c7:/home/coop>ls /home
coop  linux1
c7:/home/coop>
```

AI STUDIO

Adding and Removing Groups

↘ Adding a new group is done with **groupadd**:

↘ $ sudo /usr/sbin/groupadd anewgroup

↘ The group can be removed with:

↘ $ sudo /usr/sbin/groupdel anewgroup

↘ Adding a user to an already existing group is done with **usermod**. For example, you would first look at what groups the user already belongs to:

↘ $ groups rjsquirrel
rjsquirrel : rjsquirrel

↘ and then add the new group:

↘ $ sudo /usr/sbin/usermod -a -G anewgroup rjsquirrel

↘ $ groups rjsquirrel
rjsquirrel: rjsquirrel anewgroup

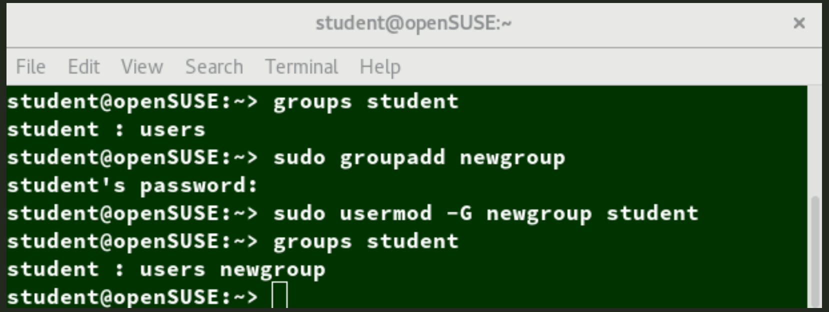
↘ These utilities update **/etc/group** as necessary. Make sure to use the -**a** option, for append, so as to avoid removing already existing groups. **groupmod** can be used to change group properties, such as the Group ID (gid) with the -**g** option or its name with then -**n** option.

↘ Removing a user from the group is somewhat trickier. The -**G** option to usermod must give a complete list of groups. Thus, if you do:

↘ $ sudo /usr/sbin/usermod -G rjsquirrel rjsquirrel

↘ $ groups rjsquirrel
rjsquirrel : rjsquirrel

↘ only the **rjsquirrel** group will be left.



```
student@openSUSE:~> groups student
student : users
student@openSUSE:~> sudo groupadd newgroup
student's password:
student@openSUSE:~> sudo usermod -G newgroup student
student@openSUSE:~> groups student
student : users newgroup
student@openSUSE:~>
```

AI STUDIO

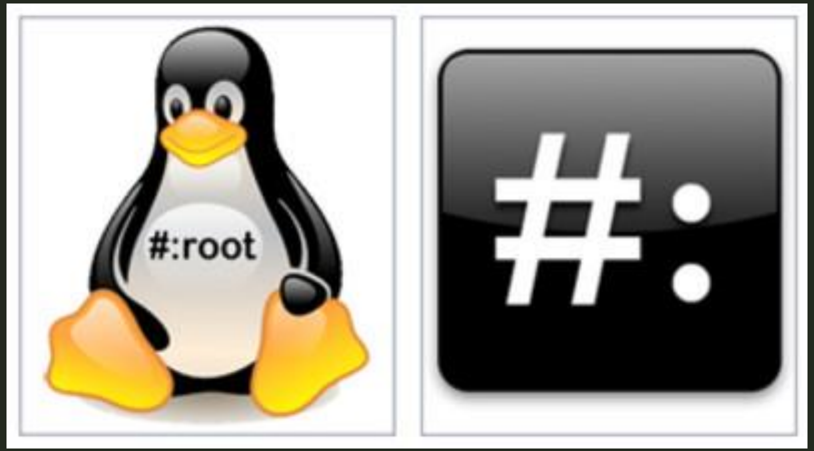
The root Account

↘ The root account is very powerful and has full access to the system. Other operating systems often call this the administrator account; in Linux, it is often called the superuser account. You must be extremely cautious before granting full root access to a user; it is rarely, if ever, justified. External attacks often consist of tricks used to elevate to the root account.

↘ However, you can use sudo to assign more limited privileges to user accounts:

↘ Only on a temporary basis

↘ Only for a specific subset of commands.



AI STUDIO

su and sudo

↘ When assigning elevated privileges, you can use the command su (switch or substitute user) to launch a new shell running as another user (you must type the password of the user you are becoming). Most often, this other user is root, and the new shell allows the use of elevated privileges until it is exited. It is almost always a bad (dangerous for both security and stability) practice to use su to become root. Resulting errors can include deletion of vital files from the system and security breaches.

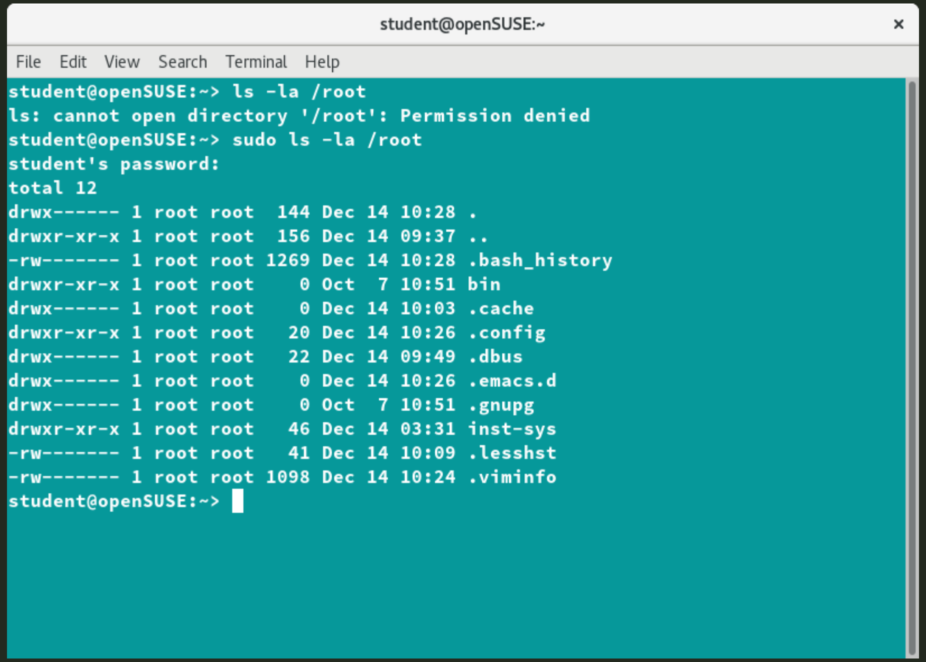
↘ Granting privileges using sudo is less dangerous and is preferred. By default, sudo must be enabled on a per-user basis. However, some distributions (such as Ubuntu) enable it by default for at least one main user, or give this as an installation option.

↘ In the Local Security Principles chapter, we will describe and compare su and sudo in detail.

AI STUDIO

Elevating to root Account

↘ To temporarily become the superuser for a series of commands, you can type su and then be prompted for the root password.
↘ To execute just one command with root privilege type sudo <command>. When the command is complete, you will return to being a normal unprivileged user.
↘ sudo configuration files are stored in the /etc/sudoers file and in the /etc/sudoers.d/ directory. By default, the sudoers.d directory is empty.



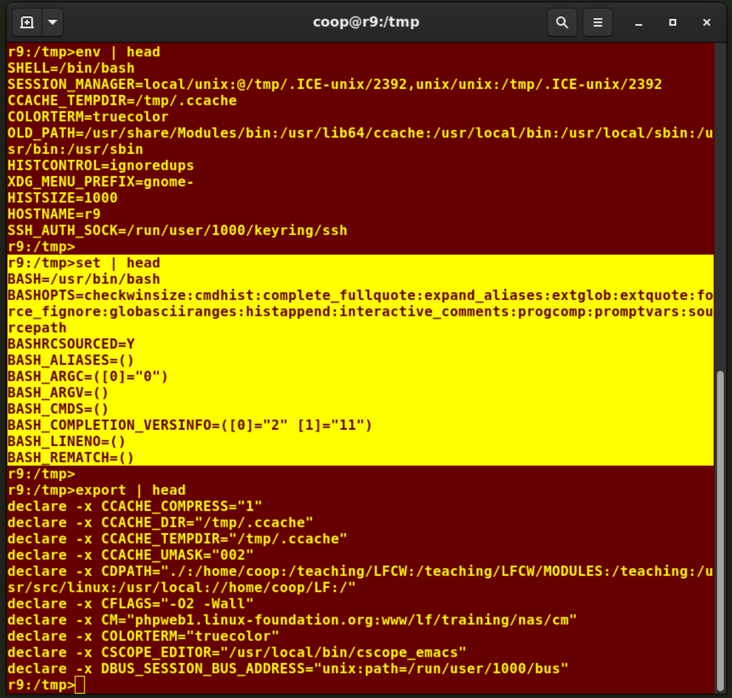
```
student@openSUSE:~> ls -la /root
ls: cannot open directory '/root': Permission denied
student@openSUSE:~> sudo ls -la /root
student's password:
total 12
drwx------ 1 root root  144 Dec 14 10:28 .
drwxr-xr-x 1 root root  156 Dec 14 09:37 ..
-rw------- 1 root root 1269 Dec 14 10:28 .bash_history
drwxr-xr-x 1 root root    0 Oct  7 10:51 bin
drwx------ 1 root root    0 Dec 14 10:03 .cache
drwxr-xr-x 1 root root   20 Dec 14 10:26 .config
drwx------ 1 root root   22 Dec 14 09:49 .dbus
drwx------ 1 root root    0 Dec 14 10:26 .emacs.d
drwx------ 1 root root    0 Oct  7 10:51 .gnupg
drwxr-xr-x 1 root root   46 Dec 14 03:31 inst-sys
-rw------- 1 root root   41 Dec 14 10:09 .lesshst
-rw------- 1 root root 1098 Dec 14 10:24 .viminfo
student@openSUSE:~>
```

AI STUDIO

Environment Variables

↘ **Environment variables** are quantities that have specific values which may be utilized by the command shell, such as **bash,** or other utilities and applications. Some environment variables are given preset values by the system (which can usually be overridden), while others are set directly by the user, either at the command line or within startup and other scripts.

↘ An environment variable is actually just a character string that contains information used by one or more applications. There are a number of ways to view the values of currently set environment variables; one can type **set, env, or export**. Depending on the state of your system, **set** may print out many more lines than the other two methods.



```
r9:/tmp>env | head
SHELL=/bin/bash
SESSION_MANAGER=local/unix:@/tmp/.ICE-unix/2392,unix/unix:/tmp/.ICE-unix/2392
CCACHE_TEMPDIR=/tmp/.ccache
COLORTERM=truecolor
OLD_PATH=/usr/share/Modules/bin:/usr/lib64/ccache:/usr/local/bin:/usr/local/sbin:/u
sr/bin:/usr/sbin
HISTCONTROL=ignoredups
XDG_MENU_PREFIX=gnome-
HISTSIZE=1000
HOSTNAME=r9
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
r9:/tmp>
r9:/tmp>set | head
BASH=/usr/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:fo
rce_fignore:globasciiranges:histappend:interactive_comments:progcomp:promptvars:sou
rcepath
BASHRCSOURCED=Y
BASH_ALIASES=()
BASH_ARGC=([0]="0")
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_VERSINFO=([0]="2" [1]="11")
BASH_LINENO=()
BASH_REMATCH=()
r9:/tmp>
r9:/tmp>export | head
declare -x CCACHE_COMPRESS="1"
declare -x CCACHE_DIR="/tmp/.ccache"
declare -x CCACHE_TEMPDIR="/tmp/.ccache"
declare -x CCACHE_UMASK="002"
declare -x CDPATH="./:/home/coop:/teaching/LFCW:/teaching/LFCW/MODULES:/teaching:/u
sr/src/linux:/usr/local://home/coop/LF:/"
declare -x CFLAGS="-O2 -Wall"
declare -x CM="phpweb1.linux-foundation.org:www/lf/training/nas/cm"
declare -x COLORTERM="truecolor"
declare -x CSCOPE_EDITOR="/usr/local/bin/cscope_emacs"
declare -x DBUS_SESSION_BUS_ADDRESS="unix:path=/run/user/1000/bus"
r9:/tmp>
```

AI STUDIO

Setting Environment Variables

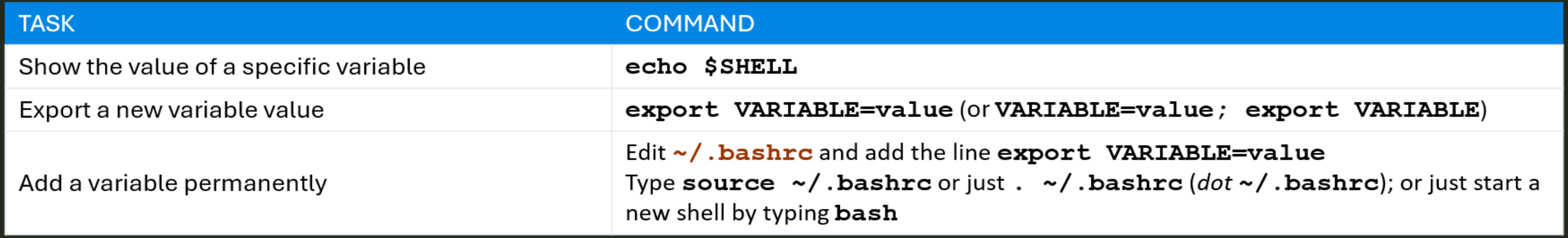
↘ By default, variables created within a script are only available to the current shell; child processes (sub-shells) will not have access to values that have been set or modified. Allowing child processes to see the values requires use of the export command.

↘ You can also set environment variables to be fed as a one shot to a command as in:

↘ **$ SDIRS="s_0*" KROOT=/lib/modules/$(uname -r)/build make modules_install**

↘ which feeds the values of the SDIRS and KROOT environment variables to the command make

modules_install.



TASK	COMMAND
Show the value of a specific variable	**`echo $SHELL`**
Export a new variable value	**`export VARIABLE=value`** (or **`VARIABLE=value; export VARIABLE`**)
Add a variable permanently	Edit **`~/.bashrc`** and add the line **`export VARIABLE=value`** Type **`source ~/.bashrc`** or just **`. ~/.bashrc`** (*dot* **`~/.bashrc`**); or just start a new shell by typing **`bash`**

AI STUDIO

The HOME Variable

↘ HOME is an environment variable that represents the home (or login) directory of the user. **cd** without arguments will change the current working directory to the value of **HOME.** Note the tilde character (~) is often used as an abbreviation for **$HOME**. Thus, **cd $HOME** and cd ~ are completely equivalent statements.

COMMAND	EXPLANATION

`$ echo $HOME`
`/home/student`
`$ cd /bin`
	Show the value of the **`HOME`** environment variable, then change directory (**`cd`**) to `/bin`.

`$ pwd`
`/bin`
	Where are we? Use print (or present) working directory (**`pwd`**) to find out. As expected, `/bin`.

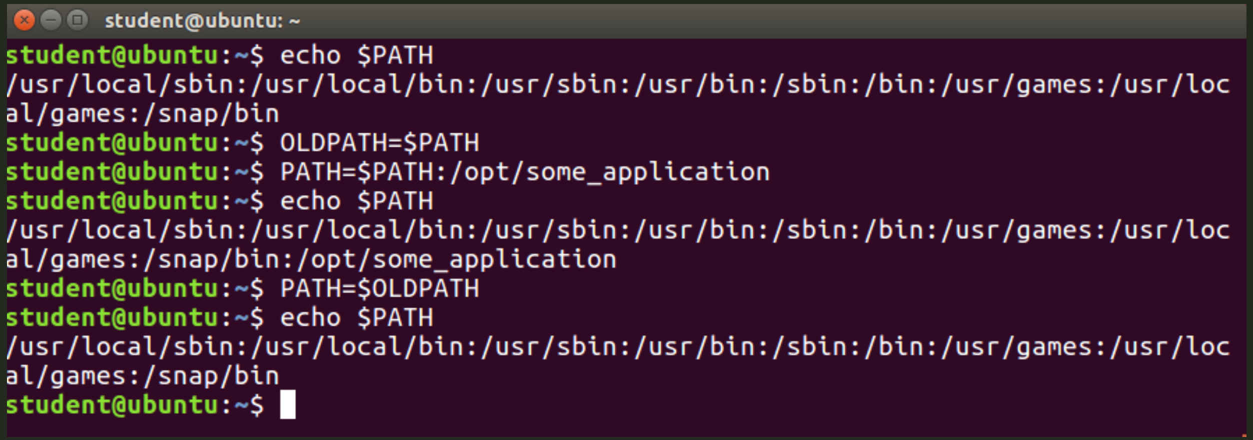
`$ cd`	Change directory without an argument...

`$ pwd`
`/home/student`
	...takes us back to **`HOME`**, as you can now see.

AI STUDIO

The PATH Variable

↘ PATH is an ordered list of directories (the path) which is scanned when a command is given to find the appropriate program or script to run. Each directory in the path is separated by colons (:). A null (empty) directory name (or ./) indicates the current directory at any given time.
↘ :path1:path2
↘ path1::path2
↘ In the example :path1:path2, there is a null directory before the first colon (:). Similarly, for path1::path2 there is a null directory between path1 and path2.
↘ To prefix a private bin directory to your path:
↘ $ export PATH=$HOME/bin:$PATH
$ echo $PATH
/home/student/bin:/usr/local/bin:/usr/bin:/bin/usr


student@ubuntu: ~

```
student@ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin
student@ubuntu:~$ OLDPATH=$PATH
student@ubuntu:~$ PATH=$PATH:/opt/some_application
student@ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin:/opt/some_application
student@ubuntu:~$ PATH=$OLDPATH
student@ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin
student@ubuntu:~$ 
```

The SHELL Variable

↘ The environment variable **SHELL** points to the user's default command shell (the program that is handling whatever you type in a command window, usually bash) and contains the full pathname to the shell:

↘
```
$ echo $SHELL
/bin/bash
$
```

AI STUDIO

The PS1 Variable and the Command Line Prompt

↘ Prompt Statement (PS) is used to customize your prompt string in your terminal windows to display the information you want.

↘ PS1 is the primary prompt variable which controls what your command line prompt looks like. The following special characters can be included in PS1:

↘ \u - User name
\h - Host name
\w - Current working directory
\! - History number of this command
\d - Date

↘ They must be surrounded in single quotes when they are used, as in the following example:

↘
```
$ echo $PS1
$
$ export PS1='\u@\h:\w$ '
student@example.com:~$ # new prompt
```

↘ To revert the changes:

↘
```
student@example.com:~$ export PS1='$ '
$
```

↘ An even better practice would be to save the old prompt first and then restore, as in:

↘
```
$ OLD_PS1=$PS1
```

↘ change the prompt, and eventually change it back with:

↘
```
$ PS1=$OLD_PS1
$
```

AI STUDIO

Sample

↘ Create a small file /tmp/ls, which contains just the line:

↘ echo HELLO, this is the phony ls program.

↘ Then, make it executable by doing:

↘ $ chmod +x /tmp/ls

↘ Append /tmp to your path, so it is searched only after your usual path is considered. Type **ls** and see which program is run: /bin/ls or /tmp/ls?

↘ Pre-pend /tmp to your path, so it is searched before your usual path is considered. Once again, type **ls** and see which program is run: /bin/ls or /tmp/ls?

↘ What are the security considerations in altering the path this way?

↘ First, create the phony **ls** program using an editor or just simply doing:

↘
```
student:/tmp>echo "echo HELLO, this is the phony ls program." > /tmp/ls
student:/tmp>chmod +x /tmp/ls
```

↘ For the next two steps, it is a good idea to work in another terminal window, or just start a new shell, so the changes do not persist on later issued commands. You can start a new shell by just typing **bash**.

↘
```
student:/tmp>bash
student:/tmp>PATH=$PATH:/tmp
student:/tmp>ls /usr
bin etc games include lib lib64 libexec local sbin share src tmp
student:/tmp>exit
```

↘
```
student:/tmp>bash
student:/tmp>PATH=/tmp:$PATH
student:/tmp>ls /usr
HELLO, this is the phony ls program.
student:/tmp>exit
```

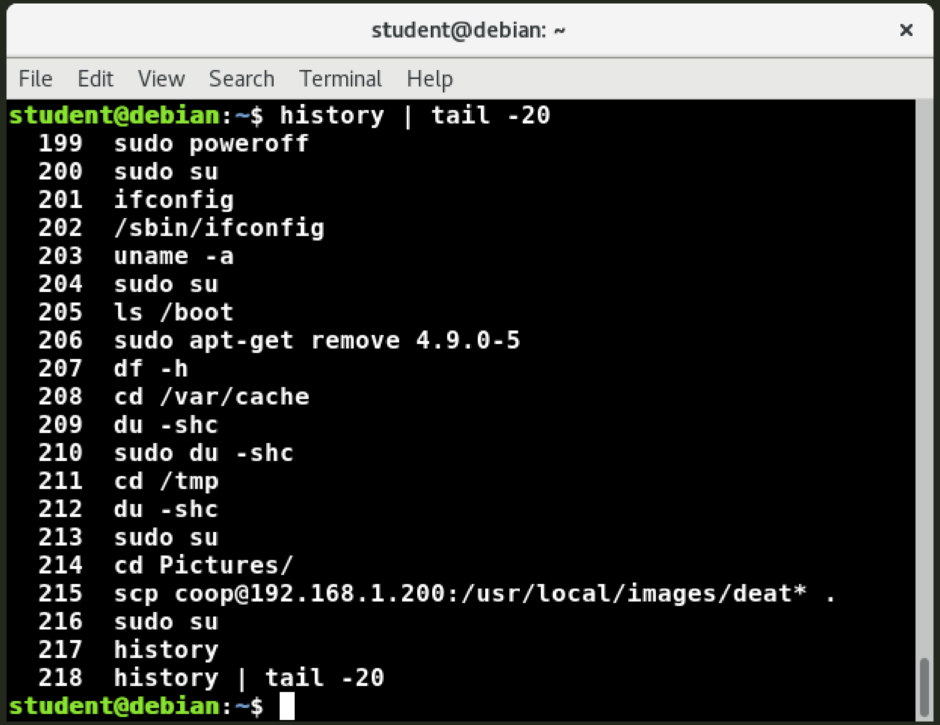
↘ Note the second form is a very dangerous thing to do, and is a trivial way to insert a **Trojan Horse** program; if someone can put a malicious program in /tmp, they can trick you into running it accidentally.

AI STUDIO

Recalling Previous Commands

↘ bash keeps track of previously entered commands and statements in a history buffer. You can recall previously used commands simply by using the Up and Down cursor keys. To view the list of previously executed commands, you can just type history at the command line.

↘ The list of commands is displayed with the most recent command appearing last in the list. This information is stored in ~/.bash_history. If you have multiple terminals open, the commands typed in each session are not saved until the session terminates.



```
student@debian:~$ history | tail -20
  199  sudo poweroff
  200  sudo su
  201  ifconfig
  202  /sbin/ifconfig
  203  uname -a
  204  sudo su
  205  ls /boot
  206  sudo apt-get remove 4.9.0-5
  207  df -h
  208  cd /var/cache
  209  du -shc
  210  sudo du -shc
  211  cd /tmp
  212  du -shc
  213  sudo su
  214  cd Pictures/
  215  scp coop@192.168.1.200:/usr/local/images/deat* .
  216  sudo su
  217  history
  218  history | tail -20
student@debian:~$
```

AI STUDIO

Using History Environment Variables

↘ Several associated environment variables can be used to get information about the history file.

↘ HISTFILE
The location of the history file.

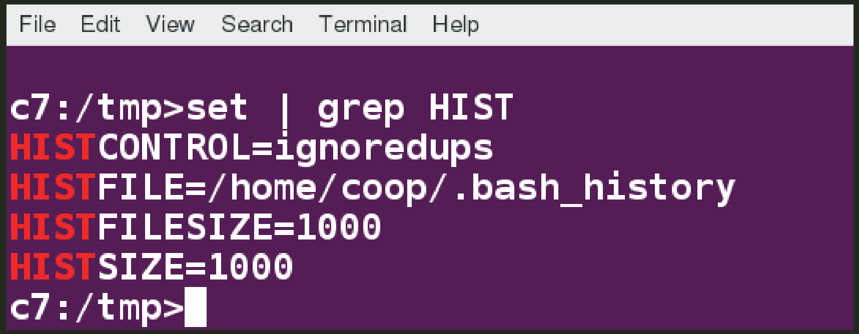
↘ HISTFILESIZE
The maximum number of lines in the history file (default 500).

↘ HISTSIZE
The maximum number of commands in the history file.

↘ HISTCONTROL
How commands are stored.

↘ HISTIGNORE
Which command lines can be unsaved.

↘ For a complete description of the use of these environment variables, see **man bash.**



```
c7:/tmp>set | grep HIST
HISTCONTROL=ignoredups
HISTFILE=/home/coop/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
c7:/tmp>
```

AI STUDIO

Finding and Using Previous Commands

↘ Specific keys to perform various tasks:

↘ If you want to recall a command in the history list, but do not want to press the arrow key repeatedly, you can press **CTRL-R** to do a reverse intelligent search.

↘ As you start typing, the search goes back in reverse order to the first command that matches the letters you have typed. By typing more successive letters, you make the match more and more specific.

↘ The following is an example of how you can use the **CTRL-R** command to search through the command history:

```
↘ $ ^R                                   (This all happens on 1 line)
(reverse-i-search)'s': sleep 1000      (Searched for 's'; matched "sleep")
$ sleep 1000                           (Pressed Enter to execute the searched command)
```

KEY	USAGE
Up/Down arrow keys	Browse through the list of commands previously executed
!! (Pronounced as bang-bang)	Execute the previous command
CTRL-R	Search previously used commands

AI STUDIO

Executing Previous Commands

↘ The table describes the syntax used to execute previously used commands:

↘ All history substitutions start with !. When typing the command: **ls –l /bin /etc /var, !$** will refer to /var, the last argument to the command.

↘ Here are more examples:

↘ $ history

↘ echo $SHELL

↘ echo $HOME

↘ echo $PS1

↘ ls -a

↘ ls -l /etc/ passwd

↘ sleep 1000

↘ history

↘ $!1 (Execute command #1 above)

```
echo $SHELL
/bin/bash
```

↘ $!sl (Execute the command beginning with "sl")

```
sleep 1000
$
```

SYNTAX	TASK
`!`	Start a history substitution
`!$`	Refer to the last argument in a line
`!n`	Refer to the nth command line
`!string`	Refer to the most recent command starting with string

AI STUDIO

Keyboard Shortcuts

↘ You can use keyboard shortcuts to perform different tasks quickly. The table lists some of these keyboard shortcuts and their uses. Note the case of the "hotkey" does not matter, e.g. doing CTRL-a is the same as doing CTRL-A .

KEYBOARD SHORTCUT	TASK
CTRL-L	Clears the screen
CTRL-S	Temporarily halt output to the terminal window
CTRL-Q	Resume output to the terminal window
CTRL-D	Exits the current shell
CTRL-Z	Puts the current process into suspended background and back to prompt
CTRL-C	Kills the current process
CTRL-H	Works the same as backspace
CTRL-A	Goes to the beginning of the line
CTRL-W	Deletes the word before the cursor
CTRL-U	Deletes from beginning of line to cursor position
CTRL-E	Goes to the end of the line
Tab	Auto-completes files, directories, and binaries

AI STUDIO

File Ownership

↘ In Linux and other UNIX-based operating systems, every file is associated with a user who is the owner. Every file is also associated with a group (a subset of all users) which has an interest in the file and certain rights, or permissions: read, write, and execute.

↘ The following utility programs involve user and group ownership and permission setting.

COMMAND	USAGE
chown	Used to change user ownership of a file or directory
chgrp	Used to change group ownership
chmod	Used to change the permissions on the file, which can be done separately for **owner**, **group** and the rest of the **world** (often named as **other**)

File Permission Modes and chmod

↘ Files have three kinds of permissions: read (**r**), write (**w**), execute (**x**). These are generally represented as in **rwx**. These permissions affect three groups of owners: user/owner (**u**), group (**g**), and others (**o**).

↘ As a result, you have the following three groups of three permissions:

↘ **rwx: rwx: rwx**
u: g: o

↘ There are a number of different ways to use **chmod**. For instance, to give the owner and others execute permission and remove the group write permission:

↘
```
$ ls -l somefile
-rw-rw-r-- 1 student student 1601 Mar 9 15:04 somefile
$ chmod uo+x,g-w somefile
$ ls -l somefile
-rwxr--r-x 1 student student 1601 Mar 9 15:04 somefile
```

↘ where **u** stands for user (owner), **o** stands for other (world), and **g** stands for group.

↘ This kind of syntax can be difficult to type and remember, so one often uses a shorthand which lets you set all the permissions in one step. This is done with a simple algorithm, and a single digit suffices to specify all three permission bits for each entity. This digit is the sum of:

↘ **4** if read permission is desired

↘ **2** if write permission is desired

↘ **1** if execute permission is desired

↘ Thus, **7** means read/write/execute, **6** means read/write, and **5** means read/execute.

↘ When you apply this to the **chmod** command, you have to give three digits for each degree of freedom, such as in:

↘
```
$ chmod 755 somefile
$ ls -l somefile
-rwxr-xr-x 1 student student 1601 Mar 9 15:04 somefile
```

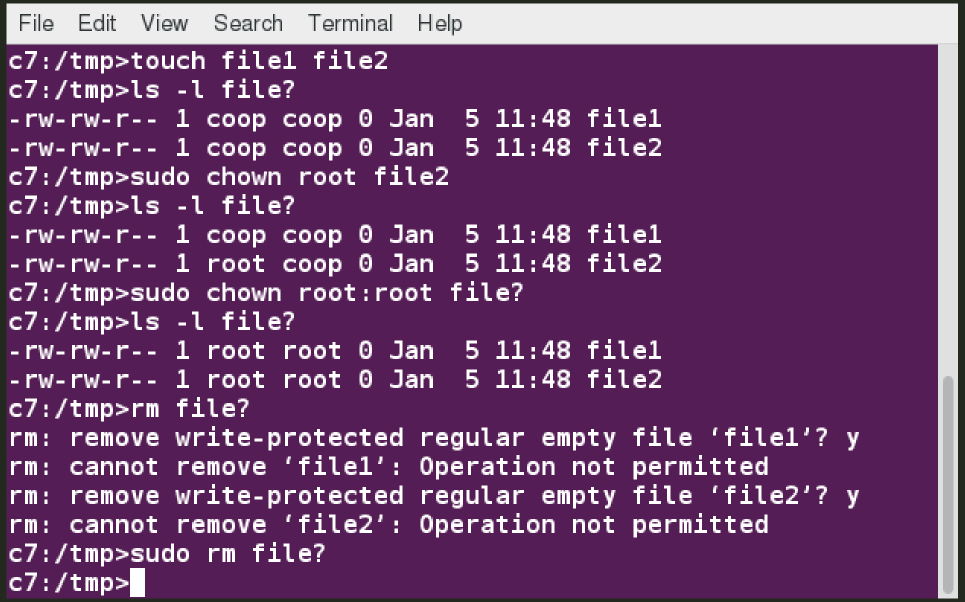
AI STUDIO

Example of chown

↘ Let's see an example of changing file ownership using chown, as shown in the screenshot to the right. First, we create two empty files using touch.

↘ Notice it requires sudo to change the owner of file2 to root. The second chown command changes both owner and group at the same time!

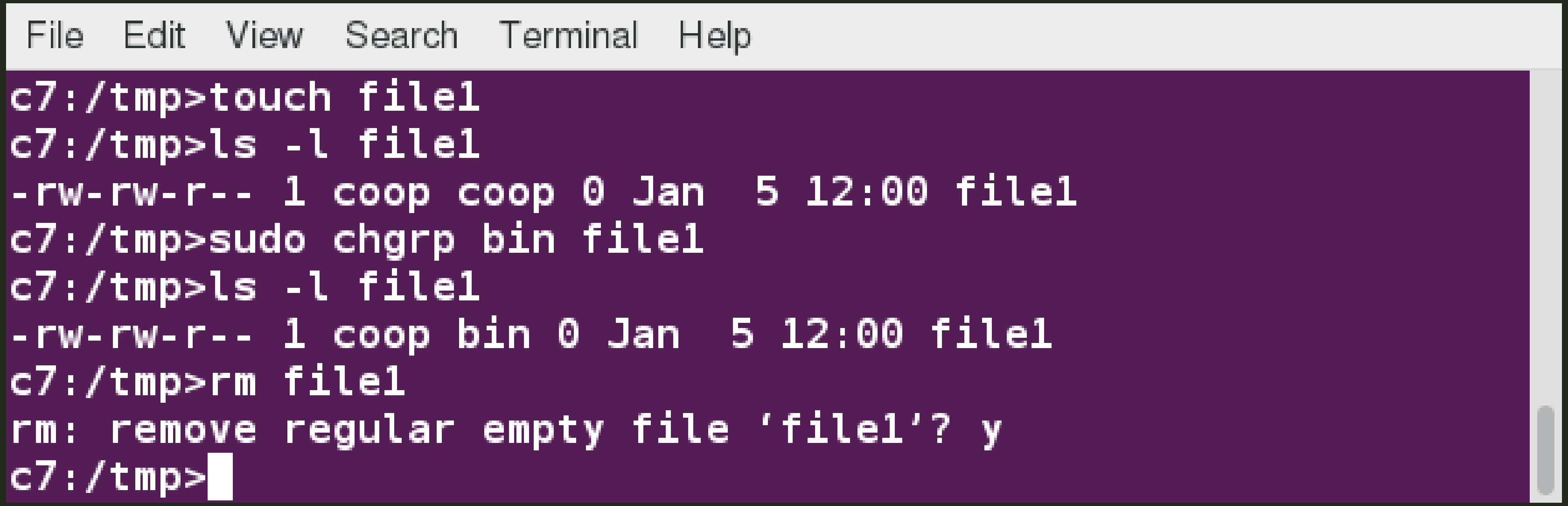
↘ Finally, only the superuser can remove the files.



```
c7:/tmp>touch file1 file2
c7:/tmp>ls -l file?
-rw-rw-r-- 1 coop coop 0 Jan  5 11:48 file1
-rw-rw-r-- 1 coop coop 0 Jan  5 11:48 file2
c7:/tmp>sudo chown root file2
c7:/tmp>ls -l file?
-rw-rw-r-- 1 coop coop 0 Jan  5 11:48 file1
-rw-rw-r-- 1 root coop 0 Jan  5 11:48 file2
c7:/tmp>sudo chown root:root file?
c7:/tmp>ls -l file?
-rw-rw-r-- 1 root root 0 Jan  5 11:48 file1
-rw-rw-r-- 1 root root 0 Jan  5 11:48 file2
c7:/tmp>rm file?
rm: remove write-protected regular empty file ‘file1’? y
rm: cannot remove ‘file1’: Operation not permitted
rm: remove write-protected regular empty file ‘file2’? y
rm: cannot remove ‘file2’: Operation not permitted
c7:/tmp>sudo rm file?
c7:/tmp>
```

Example of chgrp

↘ Now, let's see an example of changing the group ownership using chgrp:



```
c7:/tmp>touch file1
c7:/tmp>ls -l file1
-rw-rw-r-- 1 coop coop 0 Jan  5 12:00 file1
c7:/tmp>sudo chgrp bin file1
c7:/tmp>ls -l file1
-rw-rw-r-- 1 coop bin 0 Jan  5 12:00 file1
c7:/tmp>rm file1
rm: remove regular empty file 'file1'? y
c7:/tmp>
```

AI STUDIO

umask

↘ umask = User file creation mask

↘ It defines the default permissions for newly created files and directories.

↘ In Linux, when you create a new file or directory, the system starts with base permissions and then subtracts (masks out) the umask.

↘ Useful for private files and directories.

↘ 🔷 Base Permissions

↘ Files: 666 (read + write for user, group, others)

↘ Why not 777? Because execute (x) bit is not given to files by default (for security).

↘ Directories: 777 (read + write + execute for user, group, others)

↘ x is needed on directories to enter/search them.

↘ 🔷 How umask Works

↘ Formula:

↘ Final Permission = Base Permission – umask

AI STUDIO

umask usage

↘ Example 1:
umask = 022
New file: 666 - 022 = 644 → rw-r--r--
New dir: 777 - 022 = 755 → rwxr-xr-x

↘ Example 2:
umask = 077
New file: 666 - 077 = 600 → rw-------
New dir: 777 - 077 = 700 → rwx------

↘ How to Check and Set umask

↘ Check current value:

↘ Umask
0022

↘ Change temporarily (valid only in current shell):

↘ umask 027

↘ Set permanently (for a user):

↘ Add umask 027 to ~/.bashrc or ~/.profile

↘ System-wide default:

↘ /etc/profile

↘ /etc/login.defs (for new users)

AI STUDIO

Summary

↘ Linux is a multi-user system.

↘ To find the currently logged on users, you can use the who command.

↘ To find the current user ID, you can use the **whoami** command.

↘ The root account has full access to the system. It is never sensible to grant full root access to a user.

↘ You can assign root privileges to regular user accounts on a temporary basis using the sudo command.

↘ The shell program (bash) uses multiple startup files to create the user environment. Each file affects the interactive environment in a different way. /etc/profile provides the global settings.

↘ Advantages of startup files include that they customize the user's prompt, set the user's terminal type, set the command-line shortcuts and aliases, and set the default text editor, etc.

↘ An environment variable is a character string that contains data used by one or more applications. The built-in shell variables can be customized to suit your requirements.

↘ The history command recalls a list of previous commands, which can be edited and recycled.

↘ In Linux, various keyboard shortcuts can be used at the command prompt instead of long actual commands.

↘ You can customize commands by creating aliases. Adding an alias to ~/.bashrc will make it available for other shells.

↘ File permissions can be changed by typing chmod permissions filename.

↘ File ownership is changed by typing chown owner filename.

↘ File group ownership is changed by typing chgrp group filename.

↘ Define permissions for newly created files and directories by umask.

AI STUDIO

info@aistudio.com.tr

aistudio.com.tr

Thank you!