Brief Linux Command Summary

Alex. Gerbessiotis
CS Department
NJIT

AVG-24-03

September 16, 2024

1. Login and bash shell

Shell: A program that sits on top of the Operating System (kernel) and allows a user to interact with the Operating System (OS).

A user (YOU) types in **commands.**

The commands are **programs** that the shell executes through the OS on behalf of the user.

When the OS runs a program for you the **program in execution** is known as a **process**. The OS maintains about a process information about the ID of the process (Process ID or **PID** in short) and the ID of the entity (another process) that created the process in question (the Parent of the Process or **PPID**).

Every command (i.e. process) that you run has the bash shell as its parent.

Option. Every command can accept a variety of options. An option starts with the character dash (- i.e. minus) followed by, usually, single lettered options. For example -al, or **-ef**. The command and its options are separate by at least one white space character (e.g. a space or a tab).

Argument. A command includes one or more arguments separated by it or by the option or by themselves with at least one white space. For example **ls -al myfile mydir myotherfile**.

File. A file (argument) can be a regular file, a directory (a file of type directory) or anyone of a number of special files.

File argument. One denotes not only the name of the file but the location of this file. It can be described relative to the current location of the user in the user's path hierarchy or in absolute terms. Let the user's home directory be /afs/cad.njit.edu/u/a/u/auser and inside it there is a directory onedir and inside it there is a file onefile. One can describe the location of one file

(in absolute terms i.e. absolute path as) /afs/cad.njit.edu/u/a/u/auser/onedir/onefile

(relative to the home directory as) onedir/onefile

(relative to directory onedir as) onefile

Standard output: The device that displays the output of commands (usually user's terminal e.g. laptop screen)

Standard input: The device that the user uses to provide input (usually user's keyboard)

Standard error: The device used by the bash shell to display error messages asociated with the execution of a command (usually the same device as standard output).

Example 1.

% date

% whoami

% who

% hostname

The % sign is a bash shell prompt specified by a user (AUTHOR). The default one is different and it is possible to change it.

The user (YOU) types in date and then presses the Enter button. All commands must end with the pressing of the Enter button. We won't repeat this in the remainder.

By pressing the enter button, the bash shell will read your input after the prompt % and interpret it accordingly to eventually invoke a program named date that it will ask the OS system to run it on behalf of the user (YOU). A program in execution becomes a process!

The output of the command (process) date is as follows

Thu Sep 12 17:02:38 EDT 2024

Command whoami prints user's ID (at NJIT it is the myUCID), who shows all logged on users, and hostname the host ID. For example in hostname afslogin0.njit.edu, njit.edu is the domain name.

Example 2.

% ls -l myfile

% ls -l /afs/cad.njit.edu/u/a/u/auser/onedir/onefile

% ls -l ~/onedir/onefile

% ls -l ~

% ls -l /afs/cad.njit.edu/u/a/u/auser

In this example the command to be invoked is ls. It is an acronym that stands for list the files supplied as arguments to the command. We need to know where we are before supplying arguments to this command. When we login we are in the (our own) home directory. For the sake of these example home directory is the one indicated earlier and shown as **/afs/cad.njit.edu/u/a/u/auser**

Next to ls following a space (more space may appear) we see a dash/minus – followed immadiately by the letter ell. The dash indicates that an option to ls will be supplied. The option is the letter ell. It indicates a request by the user to ls to provide a long listing related to the file argument that will follow. Omitting -l a shorter output listing will be used.

After another space an argument is provided. The argument can be one or more strings of characters (digits, letters, symbols) with strings separated by a space. These are expected to be files of any type as earlier explained. The command ends with an Enter (not shown, and will not be shown or mentioned from now on).

In the first sub example we have a full path description of a file in the filesystem hierarchy.

In the second subexample we do the same using the alias ~. It stand for home directory. So if your home directory is /afs/cad.njit.edu/u/a/u/auser then ~ translates into /afs/cad.njit.edu/u/a/u/auser.

The third and fourth subexamples are identical to each other (shorhand vs longhand).

Example 3.

% ls ; date

In this example two commands are separated by a semicolon. The first command is executed first and then the second one.

Example 4.

```
% echo “How are you”          # print on standard output
% echo ~                     # print home directory
% echo $(cmd)                # print output of command
% echo $$                    # print PID of current process (bash)
% echo $?                    # print return value of last command
% echo $0                    # print name of shell (bash)
% echo $-                    # detect interactive shell
% bash                       # run an instance of bash within bash!
% exit                       # exit (terminate) running shell
% exit n                     # exit with return value integer n
% var = ``value’’            # variable var defined and assigned value
% echo $var                  # print value of var
% export var                 #make var visible to other programs
```

The echo command displays the string argument on standard output.

Example 5.

```
% cat  myfile
% more myfile
% less myfile
```

The cat command (short for catenate) displays the contents of the regular file argument on standard output. For long files one might use more or less as the output of long files is controlled and displayed one screen (e.g. laptop screen or mobaxterm screen) at a time.

Then the Spacebar (next page), button b (back/previous page), button q (quit) can be used to control the flow.

Example 6.

```
% man man

% man more

% man less
```

The command man (short for manual) gives the manual page (useful information) of the command argument on standard output. Thus the first subexample above provides info about man itself, the second about more, and so on.

Example 7.

```
% cd onedir

% /afs/cad.njit.edu/u/a/u/auser/onedir
```

The cd command (short for change directory) changes directory in both examples from the home directory into directory onedir located inside the home directory.

Example 8.

```
% history

% alias
```

A history of commands typed in the bash shell is listed. The command alias displays commands that have been renamed into something else (usually a shorter string of characters).

Example 9.

```
% ls

% ls .

% ls ~

% ls ..
```

```
% pwd
% ls -l
% ls -lt
% ls -a
% ls -lr dirname
```

The first subexample is identical to the second one. It displays all the files (short listing) of the current directory. An alias for the current directory is . i.e. the period. If you do not know your current directory command pwd (print working i.e. current directory) will display it for your. The option -l of ls displays a long listing with status, permission, timestamp, and size information. The option -t sorts the output based on access/creation/modification time. Option -a displays hidden files (usually starting with a period).

Example 10.

```
% mkdir dirname
% touch anewfile
```

Command mkdir creates a new directory inside the current directory (a.k.a. working directory). Argument dirname must not currently exist inside the current directory, whether as a regular file or a directory or any other type of a file. Otherwise an error will get displayed. If one wants to create a regular file the touch command does so by creating a zero sized file. If file anewfile already exists touch changes metadata of the file (time of last change to be the time/date of the issuing of the touch command).

Example 11.

```
% cp file1 file2
```

```
% mv file1 file2

% cp -R dir1 dir2

% mv dir3 dir4

% rm file2

% rm -rf file1

% rmdir emptydir

% rm -rf nonemptydir

% shred -u file3
```

The command cp (short of copy) makea duplicate of a file (in this case of file file1) into a new file (in this case file2). The command mv (short for mv) renames file1 into file2.

The command cp -R copies a directory dir1 into a new (previously non existing) directory dir2. This means all subdirectories and files of dir1 are copied.

The command mv applied to directories renames a dir3 into a new name, dir4.

The command rm (short for remove) removes a file. The command rmdir removes an empty directory. The system will prompt you (unless you aliased or changed default behavior) whether you indeed want to delete the file or directory.

BE EXTRA CAREFUL with the use of rm -rf. THE SYSTEM WILL NOT PROMPT YOU TO CONFIRM THE OPERATION. IT WILL SILENTLY remove file1 and nonemptydir.

Note: The command rm DOES NOT erase the contents of the file from the disk. The command shred -u will do so for HDD but not for SSD. For the latter its action would be similar to rm.

Example 12.

```
% CTRL-C
% UP-ARROW
% DOWN-ARROW
% LEFT-ARROW
% RIGHT-ARROW
% CTRL-A          #move to start of command line
% CTRL-E          #move to end
% CTRL-B          #move to previous word
% CTRL-F          #move to next word
% CTRL-K          #delete (kill) text from cursor position to end of line
% CTRL-U          #cuts to start of line
% CTRL-L          # clear terminal screen
% TAB             # autocomple command or filename
% !cmd            #repeat instance of most recent command starting with cmd
```

In a command line you may move with the command history by using the UP-ARROW or DOWN-ARROW key. This way you may reexecute a previously issued command without retyping it. For a command line LEFT-ARROW or RIGHT-ARROW but also CTRL-A or CTRL-E can help you move around the command line to edit it. Note that CTRL-A means that while you are pressing the CTRL button the A key (button) is also pressed. If you want to cancel something that your typing in do a CTRL-C.

2. Andrew File System (AFS)

When you login to a Linux machine at NJIT (referred to as an AFS machine) the default file system is a distributed file system named Andrew File System developed at CMU (Carnegie Mellon University). It is distributed in the sense that your home directory is mounted whether you login to afslogin0.njit.edu or afslogin1.njit.edu or you physically login to any one of the OSL machines of the OSL Laboratory of the 2nd floor of the GITC building. Moreover AFS is available in Windows machines in the Windows labs at NJIT.

AFS enforces access control (rights, permissions, privileges) at directory level only. For file level permissions the standard UNIX permissions are being used where read, write or execute permissions (i.e. a triad of permissions/privileges/access rights) are assigned to three entities: the user owner/creator of the file, the group of users of the user owner, and everybody else. The triad of permissions are rwx for read, write (i.e. modify) and execute (an executable file). The three entities are denotes u (user owner), g (group of user owner) and o (other users i.e. everybody else other than u and g).

Authentication on AFS and Tokens

When you login on an AFS machine your password is first checked for authentication (a Duo 2FA is an addition form of two factor authentication) to verify that you are a valid user and also a valid AFS user. You are then assigned a token or a set of tokens that allow you access to AFS files and directories. You are thus becoming a member of the groupd system:authuser of the AFS cell that is NJIT (cad or cad.njit.edu).

Note that system:anyuser is any user using any of the WorldWide cells listed in the /afs directory and this includes any AFS user who is authenticated anywhere in the world (per /afs description of what that is).

One must have tickets to get tokens. Token after they get obtain are no longer related to tickets. Tokens can expire while you have valid tickets, or tickets can expire while token are valid!

kinit: obtain ticket from Kerberos

aklog: obtain token from the Kerberos ticket

renew: get new tokens

tokens: observe tokens

unlog: destroy tokens (open file operations might get upset)

system:authuser is an authorized (say NJIT) user

system:anyuser is any web user anywhere and anyone!

Traditional Unix Privileges (permissions, access rights)

In Linux and Unix the commands to list the access rights of a file and modify them are

% ls -al	# for hidden or not files
% ls -l	# for non – hidden files
% chmod privileges filename	#to modify privileges

The command chmod can be used in a variety of way (man chmod) but for brevity we mention some but not all of them. File filename is a regular file. For directories such access permissions are ignored and AFS permissions thus supersede them. This includes permisions on files inside directories that have AFS permissions.

% chmod 777 filename	# establish rwx for all entities
% chmod 000 filename	# withdraw privilege from all entities
% chmod -w filename	# withdraw write access
% chmod +w filename	# establish write access
% chmod -r filename	# read access to file
% chmod +r filename	
% chmod -x filename	# execute privileges for scripts, executable files
% chmod +x filename	

Access control lists (ACL) : help

% fs help listacl

% fs listacl -help

% fs ls -help

% fs lista -help

Access control lists (ACL) : file quotas and other high-level commands

% fs listquota

% fs lq

% fs command -help

% fs lq -help

% fs listquota -help

% fs listquota -path PATHNAME

% fs examine -path PATHNAME

% fs whereis -path PATHNAME

% fs checkserver [-cell afscellname] [-all]

% fs listcells

The long form (listquota) or the short form (lq) used as argument for fs (short for Andrew file system) indicate the quota assigned to you the user of the system. Quota means the current size and maximum size of all (the combined) files in the filesystem that you have created and allowed to create.

The last three commands show how one can get online assistance for a specific AFS command (in fact argument to fs).

Access control lists (ACL) : View directory privileges

% fs listacl dirname

% fs la dirname

The short (la) or long (listacl) form list the privileges (protections) of the named directory.

% fs la ~

The command above shows the protections of your home directory only.

Access control lists (ACL) : Type of rights

There are two types of rights: positive and negative.

Positive rights allow other users or other groups a designated access to a directory.

Negative rights prevent access to users who would have such access otherwise.

Read, Write and Lock are file permissions; Lookup,Insert,Delete and Administer are directory permissions.

Read (r) : read any file in the designated directory and also do an ls -l

Lookup (l): list all files and obtain metadata info (e.g. do an ls, ls -ld) of a directory (but not necessarily subdirectories in it)

Insert (i): create (a.k.a insert) new files in the designated directory including new subdirectories; it does not extend to do the same in subdirectories

Delete (d): delete (a.k.a. remove) files including subdirectories from directory or move them elsewhere

Write (w): user can modify contents of files in directory and to use chmod command

Lock (k): allows user to run programs that use system call to lock files in directory

Administer (a): allows a user to change directory's ACLs.

Common aliases

read (rl)

write (rlidwk)

all (rlidwka)

none

Access control lists (ACL) : Default protections

They were created when your account was created. In Example 1 below you see what this means for NJIT. In Example 2 below you see what this means for an active Web-page at NJIT. If you have a web-page at NJIT it is hosted inside the public_html directory of you home directory.

Example 1.

% fs la ~

```
Access list for /afs/cad/u/a/u/auser is
Normal rights:
  arcsnewstaff rlidwk
  homer l
  system:administrators rlidwka
  auser rlidwka
  http l
  httpsvc.webhost03.ucs.njit.edu l
  httpsvc.webhost02.arcs.njit.edu l
  httpsvc.webhost03.arcs.njit.edu l
```

Example 2.

% fs la ~/public_html

```
Access list for /afs/cad/u/a/u/auser/public_html/ is
Normal rights:
  homer rl
  system:administrators rlidwka
  auser rlidwka
  http rl
  httpsvc.webhost03.ucs.njit.edu l
  httpsvc.webhost02.arcs.njit.edu rl
  httpsvc.webhost03.arcs.njit.edu rl
```

Access control lists (ACL) : Set directory privileges

We can set privileges to a directory directly as shown below. Such privileges are assigned to the ALLOW list i.e. the previlge list. If we expect NOT to assign those privileges we assign them to the DISALLOW list. We do this by utilizing the -negative flag. We may precede a directory name with the -dir flag and the acl list (of user and corresponding rights) with the -acl flag. Option clear clears current list before creating new one. Note that setacl DOES NOT MODIFY rights but SETs a new set of rights. Rights already assigned need to be re-assigned. A short form of -negative is -neg. Be reminded that negative rights have priority (and cancel out) regular rights (privileges).

% fs setacl dirname user rights # positive rights

% fs setacl -dir dirname -acl acces-list-par [-clear] [-negative] #general

% fs sa dirname user rights

% fs setacl dirname -negative user rights # positive rights

% fs sa dirname -negative user rights

% fs setacl dirname ptsgroup rights # -negative possible

% fs sa dirname ptsgroup rights

% fs setacl dirname <userID>@<cellname> rights # -negative possible

% fs sa dirname <userID>@<cellname> rights

% fs sa dirname -acl access-list -clear #first clear list; then create new one

Example 3.

% fs sa onedir auser none # withdraw privileges

% fs sa onedir auser newprivileges # assign select set of privileges

% fs sa onedir auser all # assign all possible privileges

% fs sa onedir auser someprivilege # assign some privileges (after a none)

% fs sa onedir -negative auser someprivilege # withdraw a privilege (after an all)

% fs sa onedir otheruser read

% fs sa onedir otheruser rl

% fs sa onedir otheruser@cs.cmu.edu read

When you use the negative command withdraw rights be careful with reinstating those rights that had been removed (withdrawn)

```
% fs sa onedir auser none                    # withdraw everything
% fs sa onedir auser rlidwk                  # reinstate all
% fs sa onedir auser all
% fs sa onedir -negative auser some          # Negative rights are also listed with fs la
% fs sa onedir -negative auser none          # to erase negative some for auser
```

Example 4.

The following command clears all prior privileges of all users and creates new privileges for just two users.

```
% fs sa -dir dirname -acl tom ALL jery rlidw -clear
```

Access control lists (ACL) : Creating pts groups

```
% pts createg auser: mygroup
% pts creategroup auser:mygroup
% pts creategroup -name groupname -owner groupowner
% pts members auser:mygroup
% pts membership [-nameorid] auser:mygroup
% pts examine [-nameorid] auser:mygroup
% pts listowned [-nameorid] auser:mygroup
% pts members auser
% pts add buser auser:mygroup
% pts adduser buser auser:mygroup
% pts adduer -user buser -group auser:mygroup
% pts remove buser auser:mygroup
```

% pts removeuser -user buser -group auser:mygroup

% pts delete -group groupname

% pts delete -nameorid userorgrouporid

% fs cleancl

Access control lists (ACL) : Copy ACL

% fs copyacl -fromdir dirname1 -todir dirname2 [-clear]

The use of -clear first clears the ACLs of dirname2 before those of dirname1 are applied.

3 Other commands

A list of commands for process management and view of system status is shown below.

Example 1.

% egrep string filename	# search contents of file filename for string
% stat filename	# info on filename
% w	# what is going on in system?
% finger	# similar to w
% ps	# process status for current login terminal
% ps -f	# PID and PPID displayed
% ps -ef | egrep myUCID	# find processes related to myUCID
% kill -9 PID	# kill (terminate) process with pid PID
% top	# list of processes (similar to ps)
% ./prx	# run program (e.g. executable file) named prx and located
	# inside the current directory!
% CTRL-Z	# while prx is running CTRL-Z suspends execution

```
% CTRL-C          # while prx is running CTRL-C terminates it
% jobs            # A list of processes in the background is shown

% bg              #  The suspended process (prx) is reinstated and
                  #  resume execution in the background
% fg              # A suspended process resume execution in the  foregound
% bg %n           # n  is  a numeric value associated with process
% fg %n           # bring process into foreground
% kill %n         #  Terminates / kill process in the background
```

A list of commands for dealing with archive programs such as tar (tape archive), zip, unzip and gzip is shown below.

Example 2.

```
% tar cvf x.tar a.txt b.txt   # Pack a.txt,b.txt into x.tar
% tar xvf x.tar               # Unpack x.tar into its contents
                                  x: extract,  v: verbose
                                  f: tar file follows (x.tar)
                                 c: create
% zip f.zip a.txt b.txt       # Pack a.txt,b.txt into f.zip
% unzip f.zip                 # Unpack f.zip

% gzip x.tar                  # Compress x.tar
% gzip -d x.tar.gz            # Uncompress x.tar.gz
```

A list of commands for dealing with command line compile tools (GCC toolchain) is shown below.

Example 3.

```
% gcc -v                          # List gcc version
% gcc pr1.c                       # compile source code file pr1.c into executable file a.out
% gcc pr1.c -o pr1                # compile source code file pr1.c into executable file pr1
% gcc pr1.c -lm -o pr1            # link also math library libmath (code has sin,cos,etc functions)
% gcc pr1.c -lpthread -o pr1      # link with pthread library instead of libmath

% ./prx                                  # run program prx from current directory
% echo $?                        # run immediately after the termination/conclusion of prx
                                 # to obtain the return value (or the main function) of prx
```