



Red Hat System Administration I

Dr. Eng. Gamal Selim

Lecture 03



Agenda

- **Chapter 5: Create, View, and Edit Text Files**
- **Chapter 6: Manage Local Users and Groups**

Chapter 5: Create, View, and Edit Text Files

CREATING, VIEWING, AND EDITING TEXT FILES

GOAL

To create, view, and edit text files from command output or in an editor.

OBJECTIVES

- Redirect the text output of a program to a file or to another program.
- Edit existing text files and create new files from the shell prompt with a text editor.
- Copy text from a graphical window to a text file using a text editor running in the graphical environment.

SECTIONS

- Redirecting Output to a File or Program (and Practice)
- Editing Text Files from the Shell Prompt (and Practice)
- Editing Text Files with a Graphical Editor (and Practice)

REDIRECTING OUTPUT TO A FILE OR PROGRAM

OBJECTIVES

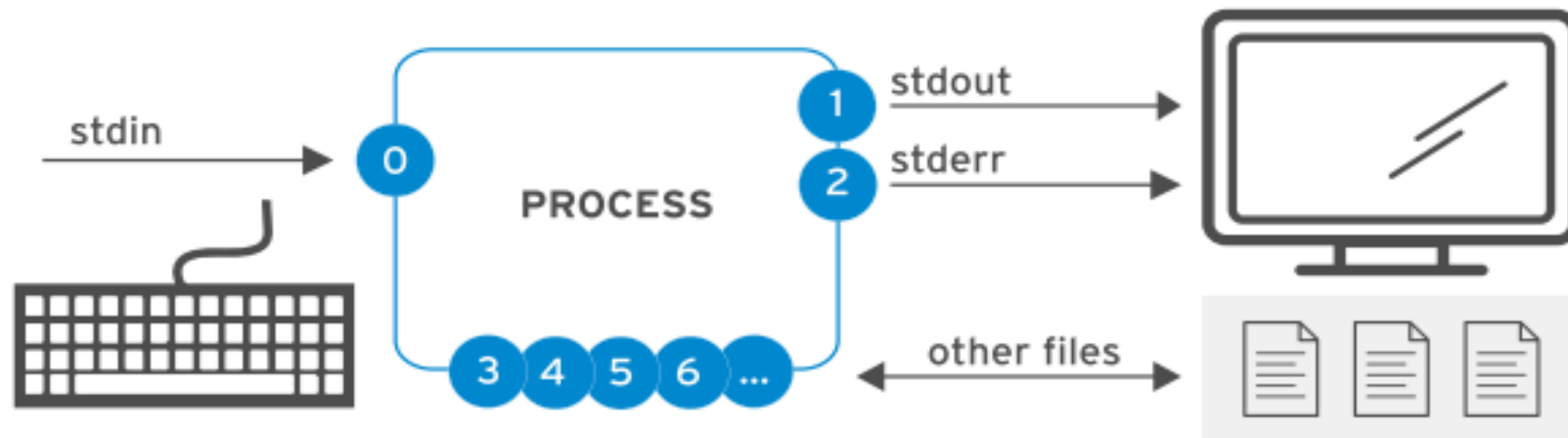
After completing this section, students should be able to:

- Describe the technical terms standard input, standard output, and standard error.
- Use redirection characters to control output to files.
- Use piping to control output to other programs.

STANDARD INPUT, STANDARD OUTPUT, AND STANDARD ERROR

A running program, or *process*, needs to read input from somewhere and write output to the screen or to files. A command run from the shell prompt normally reads its input from the keyboard and sends its output to its terminal window.

A process uses numbered channels called *file descriptors* to get input and send output. All processes will have at least three file descriptors to start with. *Standard input* (channel 0) reads input from the keyboard. *Standard output* (channel 1) sends normal output to the terminal. *Standard error* (channel 2) sends error messages to the terminal. If a program opens separate connections to other files, it may use higher-numbered file descriptors.



Channels (File Descriptors)

NUMBER	CHANNEL NAME	DESCRIPTION	DEFAULT CONNECTION	USAGE
0	stdin	Standard input	Keyboard	read only
1	stdout	Standard output	Terminal	write only
2	stderr	Standard error	Terminal	write only
3+	<i>filename</i>	Other files	<i>none</i>	read and/or write

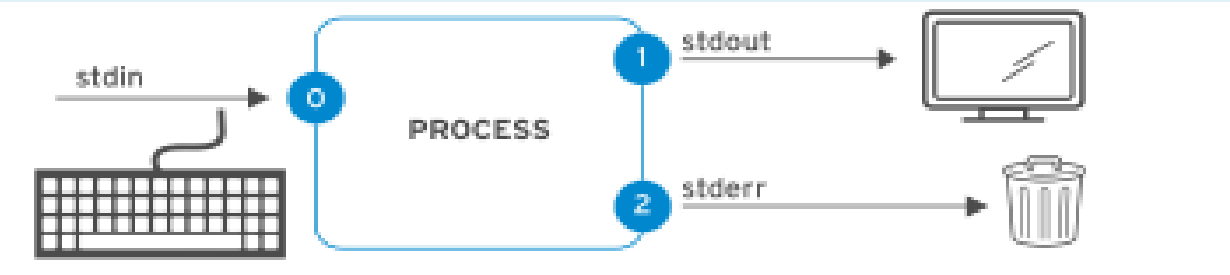
REDIRECTING OUTPUT TO A FILE

I/O *redirection* replaces the default channel destinations with file names representing either output files or devices. Using redirection, process output and error messages normally sent to the terminal window can be captured as file contents, sent to a device, or discarded.

Redirecting **stdout** suppresses process output from appearing on the terminal. As seen in the following table, redirecting *only stdout* does not suppress **stderr** error messages from displaying on the terminal. If the file does not exist, it will be created. If the file does exist and the redirection is not one that appends to the file, the file's contents will be overwritten. The special file **/dev/null** quietly discards channel output redirected to it and is always an empty file.

Output Redirection Operators

USAGE	EXPLANATION	VISUAL AID
> file	redirect stdout to overwrite a file	
>> file	redirect stdout to append to a file	
2> file	redirect stderr to overwrite a file	

2>/dev/null	discard stderr error messages by redirecting to /dev/null	 The diagram illustrates a process box with three numbered ports: 0 (stdin), 1 (stdout), and 2 (stderr). An arrow labeled 'stdin' points from a keyboard icon to port 0. An arrow labeled 'stdout' points from port 1 to a monitor icon. An arrow labeled 'stderr' points from port 2 to a trash can icon.
>file 2>&1	redirect stdout and stderr to overwrite the same file	 The diagram shows a process box with ports 0, 1, and 2. 'stdin' from a keyboard goes to port 0. Both 'stdout' (port 1) and 'stderr' (port 2) have arrows pointing to a single document icon representing a file.
>>file 2>&1	redirect stdout and stderr to append to the same file	 The diagram is similar to the previous one, but the document icon representing the file has two downward-pointing arrows below it, indicating that the output is being appended to the file.

Because of this, some people prefer to use the merging redirection operators:

&>file	instead of	>file 2>&1
----------------------	---------------	-----------------------------

&>>file	instead of	>>file 2>&1 (in Bash 4 / RHEL 6 and later)
--------------------------	---------------	--

Examples for output redirection

Many routine administration tasks are simplified by using redirection. Use the previous table to assist while considering the following examples:

- Save a timestamp for later reference.

```
[student@desktopX ~]$date > /tmp/saved-timestamp
```

- Copy the last 100 lines from a log file to another file.

```
[student@desktopX ~]$tail -n 100 /var/log/dmesg > /tmp/last-100-boot-messages
```

- Concatenate four files into one.

```
[student@desktopX ~]$cat file1 file2 file3 file4 > /tmp/all-four-in-one
```

- List the home directory's hidden and regular file names into a file.

```
[student@desktopX ~]$ls -a > /tmp/my-file-names
```

CONSTRUCTING PIPELINES

A *pipeline* is a sequence of one or more commands separated by `|`, the *pipe* character. A pipe connects the standard output of the first command to the standard input of the next command.

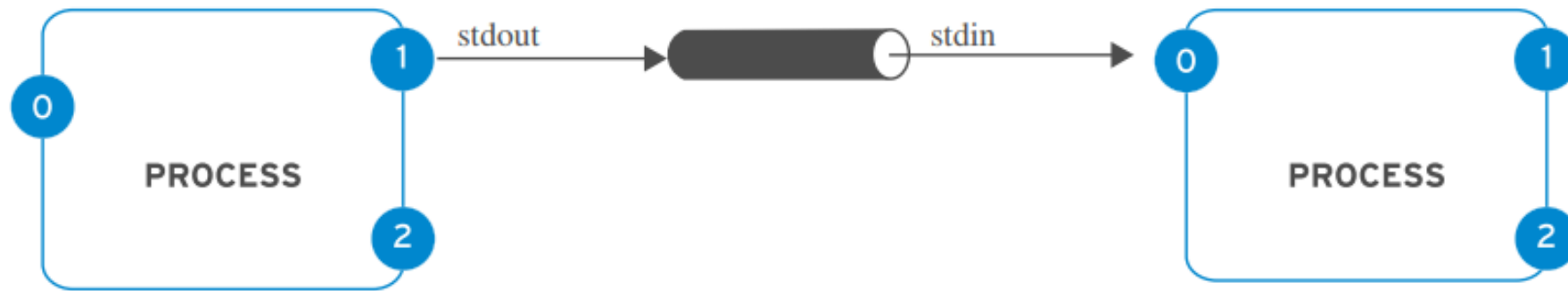


Figure 4.8: Process I/O piping

Pipelines allow the output of a process to be manipulated and formatted by other processes before it is output to the terminal. One useful mental image is to imagine that data is "flowing" through the pipeline from one process to another, being altered in slight ways by each command in the pipeline through which it passes.

Pipeline examples

This example takes the output of the **ls** command and uses **less** to display it on the terminal one screen at a time.

```
[student@desktopX ~]$ls -l /usr/bin | less
```

The output of the **ls** command is piped to **wc -l**, which counts the number of lines received from **ls** and prints that to the terminal.

```
[student@desktopX ~]$ls | wc -l
```

In this pipeline, **head** will output the first 10 lines of output from **ls -t**, with the final result redirected to a file.

```
[student@desktopX ~]$ls -t | head -n 10 > /tmp/ten-last-changed-files
```

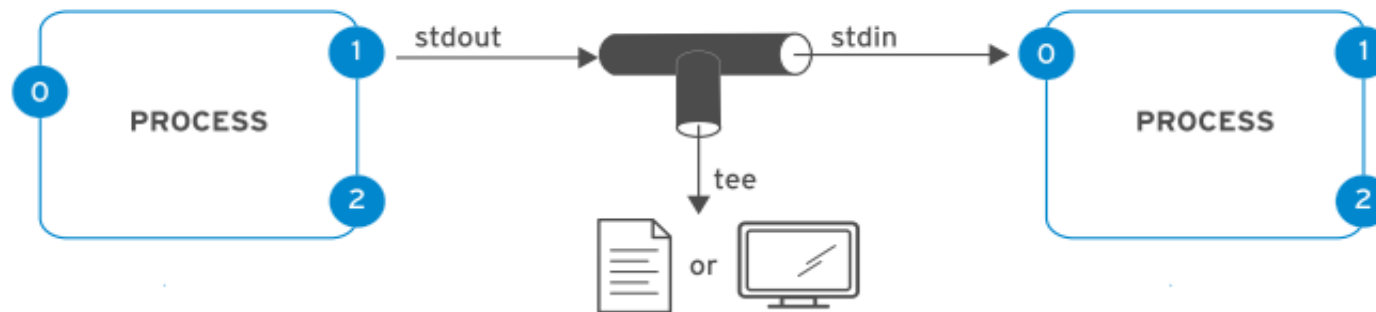
Pipelines, redirection, and tee

When redirection is combined with a pipeline, the shell first sets up the entire pipeline, then it redirects input/output. What this means is that if output redirection is used in the *middle* of a pipeline, the output will go to the file and not to the next command in the pipeline.

In this example, the output of the **ls** command will go to the file, and **less** will display nothing on the terminal.

```
[student@desktopX ~]$ls > /tmp/saved-output | less
```

The **tee** command is used to work around this. In a pipeline, **tee** will copy its standard input to its standard output and will also redirect its standard output to the files named as arguments to the command. If data is imagined to be like water flowing through a pipeline, **tee** can be visualized as a "T" joint in the pipe which directs output in two directions.



EDITING TEXT FILES FROM THE SHELL PROMPT

OBJECTIVES

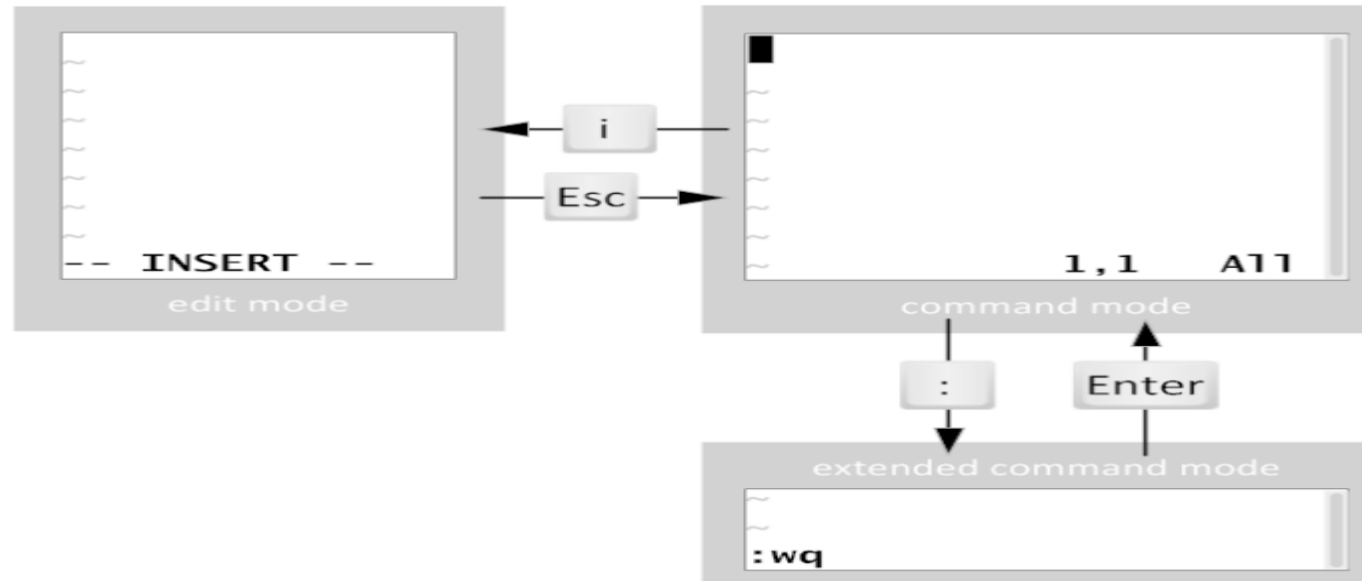
After completing this section, students should be able to:

- Create new files and edit existing text files from the shell prompt.
- Navigate within an editor to effectively accomplish editing tasks.

EDITING FILES WITH VIM

A key design principle of Linux is that information is stored in text-based files. Text files include both *flat files* with rows of similar information, such as configuration files in **/etc**, and *Extensible Markup Language (XML)* files, which define data structure through text tags, seen in application configuration files throughout both **/etc** and **/usr**. The advantage of text files is that they can be moved or shared between systems without requiring conversion, and 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.



Insert & Replace

insert mode

i	insert text before the cursor
a	insert text after the cursor
I	insert text at the beginning of the current line
A	insert text at the end of the current line
o	insert text in a new line below the cursor
O	insert text in a new line above the cursor
r	replace a character at the cursor position
R	replace characters starting at the cursor position

Undo, Delete, and Copy

edit commands

u	undo last change
CTRL+r	redo last undo change
mA	set mark A to current position. Marks can be the letter [a-z] and [A-Z]
y'A	yank (copy) from current line to mark A
d'A	delete from current line to mark A
P	put the buffer content before the cursor
p	put the buffer content after the cursor
x	delete character at cursor position
dW	delete first word after cursor position
d\$ or D	delete from cursor position to end of line
dd	delete current line
J	join current line with following line

Search & Substitute

patterns

<code>/string</code>	search forward for string
<code>?string</code>	search backward for string
<code>n</code>	repeat the last search
<code>:\$s/str1/ str2/gc</code>	search and substitute str1 by str2 from the current line to the end of file. A line number can be specified before the comma ",". The "c" ask for confirmation before each substitution
<code>:%s/str1/ str2/g</code>	replace all str1 by str2 in all the file without prompting for confirmation
<code>:%s/str1/ str2/</code>	replace str1 by str2 on the first occurrence in each line of the file

Ex Commands

vi features

<code>:q</code>	quit current open file
<code>:q!</code>	force to quit without saving changes
<code>:w</code>	save file
<code>:w file</code>	save file as file
<code>:wq!</code>	force to save file then quit
<code>:x,yw file</code>	write from line x to line y into file
<code>:w » file</code>	append buffer to file

EDITING TEXT FILES WITH A GRAPHICAL EDITOR

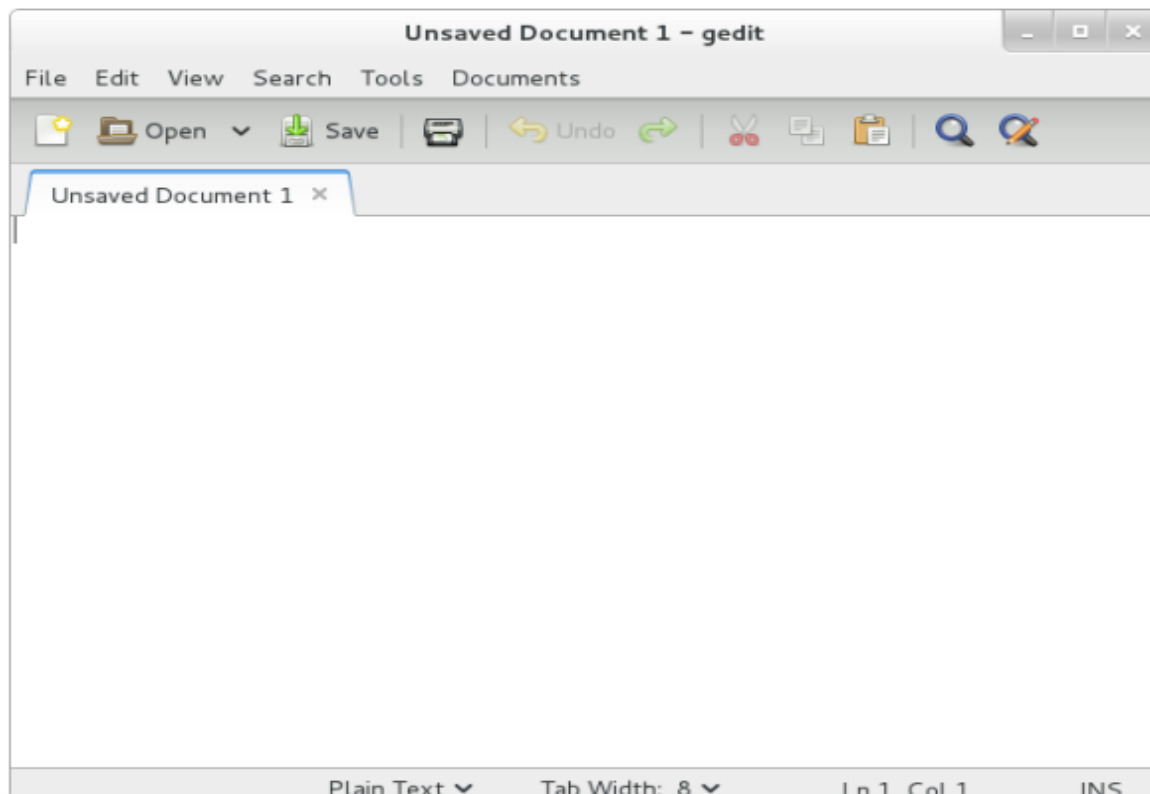
OBJECTIVES

After completing this section, students should be able to:

- Edit text files with gedit.
- Copy text between graphical windows.

EDITING FILES WITH GEDIT

The **gedit** application is a full-featured text editor for the GNOME desktop environment. Launch **gedit** by selecting **Applications** → **Accessories** → **gedit** from the GNOME menu. Like other graphical applications, **gedit** can be started without navigating the menu. Press **Alt+F2** to open the Enter a Command dialog box. Type **gedit** and press **Enter**.



Displaying commands

cat / more / less

head / tail

cut

sort

```
# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
...
```

The head Command

To review the start of a file, use the head command:

```
# head -5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

The tail Command

When reviewing log files, you are probably interested in only the last few entries of that file. You can use the tail command to look at the end of the file.

```
# tail -5 /etc/passwd
systemd-network:x:995:991:systemd Network Management:/:/sbin/nologin
geoclue:x:994:990:User for geoclue:/var/lib/geoclue:/sbin/nologin
cloud_user:x:1002:1003::/home/cloud_user:/bin/bash
ssm-user:x:1003:1004::/home/ssm-user:/bin/bash
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
```

Cut

cd pcfb/examples/

\$cut -f 1,3 Thal*.txt

returns columns 1 and 3 delimited by tabs

\$cut -f 1-3 Thal*.txt

returns columns 1 to 3 delimited by tabs

\$cut -c 16-20,30 Thal*.txt

returns characters 16 to 20 and 30 from each line

\$grep ">" FPexamples.fta | cut -c 2-11

prints out the gene names

\$head ctd.txt | cut -f 5,7 -d ","

returns columns 5 and 7. These are delimited by , in the original file and in the output.

To sort the file by the 5th field (the comment), where fields are separated by the colon, use the following command:

```
# sort -t: -k5 /etc/passwd
tss:x:59:59:Account used by the trousers package to sandbox the tcsd
adm:x:3:4:adm:/var/adm:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/st
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nol
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
halt:x:7:0:halt:/sbin:/sbin/halt
...
```

End of Chapter 5

Chapter 6: Manage Local Users and Groups

MANAGING LOCAL LINUX USERS AND GROUPS

GOAL

To manage local Linux users and groups and administer local password policies.

OBJECTIVES

- Explain the role of users and groups on a Linux system and how they are understood by the computer.
- Run commands as the superuser to administer a Linux system.
- Create, modify, lock, and delete locally defined user accounts.
- Create, modify, and delete locally defined group accounts.
- Lock accounts manually or by setting a password-aging policy in the shadow password file.

SECTIONS

- Users and Groups (and Practice)
- Gaining Superuser Access (and Practice)
- Managing Local User Accounts (and Practice)
- Managing Local Group Accounts (and Practice)
- Managing User Passwords (and Practice)

USERS AND GROUPS

OBJECTIVES

After completing this section, students should be able to explain the role of users and groups on a Linux system and how they are understood by the computer.

WHAT IS A USER?

Every process (running program) on the system runs as a particular user. Every file is owned by a particular user. Access to files and directories are restricted by user. The user associated with a running process determines the files and directories accessible to that process.

The **id** command is used to show information about the current logged-in user. Basic information about another user can also be requested by passing in the username of that user as the first argument to the **id** command.

```
[student@desktopX ~]$ id
uid=1000(student) gid=1000(student) groups=1000(student),10(wheel)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

To view the user associated with a file or directory, use the **ls -l** command. The third column shows the username:

```
[student@serverX ~]$ ls -l /tmp
drwx-----. 2 gdm      gdm      4096 Jan 24 13:05 orbit-gdm
drwx-----. 2 student  student 4096 Jan 25 20:40 orbit-student
-rw-r--r--. 1 root     root     23574 Jan 24 13:05 postconf
```

To view process information, use the **ps** command. The default is to show only processes in the current shell. Add the **a** option to view all processes with a terminal. To view the user associated with a process, include the **u** option. The first column shows the username:

```
[student@serverX ~]$ ps au
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	428	0.0	0.7	152768	14400	tty1	Ss+	Feb03	0:04	/usr/bin/Xorg
root	511	0.0	0.0	110012	812	ttyS0	Ss+	Feb03	0:00	/sbin/agetty
root	1805	0.0	0.1	116040	2580	pts/0	Ss	Feb03	0:00	-bash
root	2109	0.0	0.1	178468	2200	pts/0	S	Feb03	0:00	su - student
student	2110	0.0	0.1	116168	2864	pts/0	S	Feb03	0:00	-bash
student	3690	0.0	0.0	123368	1300	pts/0	R+	11:42	0:00	ps au

The output of the previous commands displays users by name, but internally the operating system tracks users by a *UID number*. The mapping of names to numbers is defined in databases of account information. By default, systems use a simple "flat file," the **/etc/passwd** file, to store information about local users. The format of **/etc/passwd** follows (seven colon-separated fields):

```
1 username: 2 password: 3 UID: 4 GID: 5 GECOS: 6 /home/dir: 7 shell
```

- 1 *username* is a mapping of a UID to a name for the benefit of human users.
- 2 *password* is where, historically, passwords were kept in an encrypted format. Today, they are stored in a separate file called **/etc/shadow**.
- 3 *UID* is a user ID, a number that identifies the user at the most fundamental level.
- 4 *GID* is the user's primary group ID number. Groups will be discussed in a moment.
- 5 *GECOS* field is arbitrary text, which usually includes the user's real name.
- 6 */home/dir* is the location of the user's personal data and configuration files.
- 7 *shell* is a program that runs as the user logs in. For a regular user, this is normally the program that provides the user's command line prompt.

WHAT IS A GROUP?

Like users, groups have a name and a number (GID). Local groups are defined in **/etc/group**.

Primary groups

- Every user has exactly one *primary group*.
- For local users, the primary group is defined by the GID number of the group listed in the fourth field of **/etc/passwd**.
- Normally, the primary group owns new files created by the user.
- Normally, the primary group of a newly created user is a newly created group with the same name as the user. The user is the only member of this *User Private Group* (UPG).

Supplementary groups

- Users may be a member of zero or more *supplementary groups*.
- The users that are supplementary members of local groups are listed in the last field of the group's entry in **/etc/group**. For local groups, user membership is determined by a comma-separated list of users found in the last field of the group's entry in **/etc/group**:

```
groupname:password:GID:list, of, users, in, this, group
```

- Supplementary group membership is used to help ensure that users have access permissions to files and other resources on the system.

GAINING SUPERUSER ACCESS

OBJECTIVES

After completing this section, students should be able to run commands as the superuser to administer a Linux system.

THE **root** USER

Most operating systems have some sort of *superuser*, a user that has all power over the system. This user in Red Hat Enterprise Linux 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. In order to perform tasks such as installing or removing software and to manage system files and directories, a user must escalate privileges to the **root** user.

Caution !

This unlimited privilege, however, comes with responsibility. **root** has unlimited power to damage the system: remove files and directories, remove user accounts, add backdoors, etc. If the **root** account is compromised, someone else would have administrative control of the system. Throughout this course, administrators will be encouraged to log in as a normal user and escalate privileges to **root** only when needed.

The **root** account on Linux is roughly equivalent to the local Administrator account on Windows. In Linux, most system administrators log into an unprivileged user account and use various tools to temporarily gain root privileges.

SWITCHING USERS WITH **su**

The **su** command allows a user to switch to a different user account. If a username is not specified, the *root* account is implied. When invoked as a regular user, a prompt will display asking for the

RUNNING COMMANDS AS *root* WITH **sudo**

Fundamentally, Linux implements a very coarse-grained permissions model: *root* can do everything, other users can do nothing (systems-related). The common solution previously discussed is to allow standard users to temporarily “become *root*” using the **su** command. The disadvantage is that while acting as *root*, all the privileges (and responsibilities) of *root* are granted. Not only can the user restart the web server, but they can also remove the entire **/etc** directory. Additionally, all users requiring superuser privilege in this manner must know the **root** password.

The **sudo** command allows a user to be permitted to run a command as *root*, or as another user, based on settings in the **/etc/sudoers** file. Unlike other tools such as **su**, **sudo** requires users to enter their own password for authentication, not the password of the account they are trying to access. This allows an administrator to hand out fine-grained permissions to users to delegate system administration tasks, without having to hand out the *root* password.

For example, when **sudo** has been configured to allow the user *student* to run the command **usermod** as *root*, *student* could run the following command to lock a user account:

```
[student@serverX ~]$ sudo usermod -L username  
[sudo] password for student: password
```

To enable similar behavior on earlier versions of Red Hat Enterprise Linux, use **visudo** to edit the configuration file and uncomment the line allowing the group **wheel** to run all commands.

```
[root@desktopX ~]# cat /etc/sudoers
...Output omitted...
## Allows people in group wheel to run all commands
%wheel          ALL=(ALL)          ALL

## Same thing without a password
# %wheel  ALL=(ALL)          NOPASSWD: ALL
...Output omitted...
```

MANAGING LOCAL USER ACCOUNTS

OBJECTIVES

After completing this section, students should be able to create, modify, lock, and delete locally defined user accounts.

MANAGING LOCAL USERS

A number of command-line tools can be used to manage local user accounts.

useradd creates users

- **useradd *username*** sets reasonable defaults for all fields in **/etc/passwd** when run without options. The **useradd** command does not set any valid password by default, and the user cannot log in until a password is set.
- **useradd --help** will display the basic options that can be used to override the defaults. In most cases, the same options can be used with the **usermod** command to modify an existing user.
- Some defaults, such as the range of valid UID numbers and default password aging rules, are read from the **/etc/login.defs** file. Values in this file are only used when creating new users. A change to this file will not have an effect on any existing users.

UID ranges

Specific UID numbers and ranges of numbers are used for specific purposes by Red Hat Enterprise 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 Red Hat.
- *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 just the resources they need to function.
- *UID 1000+* is the range available for assignment to regular users.

usermod modifies existing users

- **usermod --help** will display the basic options that can be used to modify an account. Some common options include:

USERMOD OPTIONS:	
-c, --comment COMMENT	Add a value, such as a full name, to the GECOS field.
-g, --gid GROUP	Specify the primary group for the user account.
-G, --groups GROUPS	Specify a list of supplementary groups for the user account.
-a, --append	Used with the -G option to append the user to the supplemental groups mentioned without removing the user from other groups.
-d, --home HOME_DIR	Specify a new home directory for the user account.
-m, --move-home	Move a user home directory to a new location. Must be used with the -d option.
-s, --shell SHELL	Specify a new login shell for the user account.
-L, --lock	Lock a user account.
-U, --unlock	Unlock a user account.

userdel deletes users

- **userdel *username*** removes the user from **/etc/passwd**, but leaves the home directory intact by default.
- **userdel -r *username*** removes the user and the user's home directory.

```
[root@serverX ~]# useradd prince
[root@serverX ~]# ls -l /home
drwx----- . 3 prince prince 74 Feb 4 15:22 prince
[root@serverX ~]# userdel prince
[root@serverX ~]# ls -l /home
drwx----- . 3 1000 1000 74 Feb 4 15:22 prince
[root@serverX ~]# useradd bob
[root@serverX ~]# ls -l /home
drwx----- . 3 bob bob 74 Feb 4 15:23 bob
drwx----- . 3 bob bob 74 Feb 4 15:22 prince
```

passwd sets passwords

- **passwd *username*** can be used to either set the user's initial password or change that user's password.
- The *root* user can set a password to any value. A message will be displayed if the password does not meet the minimum recommended criteria, but is followed by a prompt to retype the new password and all tokens are updated successfully.

```
[root@serverX ~]# passwd student
Changing password for user student.
New password: redhat123
BAD PASSWORD: The password fails the dictionary check - it is based on a
dictionary word
Retype new password: redhat123
passwd: all authentication tokens updated successfully.
```

MANAGING LOCAL GROUP ACCOUNTS

OBJECTIVES

After completing this section, students should be able to create, modify, and delete locally defined group accounts.

MANAGING SUPPLEMENTARY GROUPS

A group must exist before a user can be added to that group. Several command-line tools are used to manage local group accounts.

groupadd creates groups

- **groupadd** *groupname* without options uses the next available GID from the range specified in the **/etc/login.defs** file.
- The **-g** *GID* option is used to specify a specific GID.

```
[student@serverX ~]$ sudo groupadd -g 5000 ateam
```

groupmod modifies existing groups

- The **groupmod** command is used to change a group name to a GID mapping. The **-n** option is used to specify a new name.

```
[student@serverX ~]$ sudo groupmod -n javaapp appusers
```

- The **-g** option is used to specify a new GID.

```
[student@serverX ~]$ sudo groupmod -g 6000 ateam
```

groupdel deletes a group

- The **groupdel** command will remove a group.

```
[student@serverX ~]$ sudo groupdel javaapp
```

usermod alters group membership

- The membership of a group is controlled with user management. Change a user's primary group with **usermod -g *groupname***.

```
[student@serverX ~]$ sudo usermod -g student student
```

- Add a user to a supplementary group with **usermod -aG *groupname username***.

```
[student@serverX ~]$ sudo usermod -aG wheel elvis
```

MANAGING USER PASSWORDS

OBJECTIVES

After completing this section, students should be able to lock accounts manually or by setting a password-aging policy in the shadow password file.

SHADOW PASSWORDS AND PASSWORD POLICY

In the distant past, encrypted passwords were stored in the world-readable `/etc/passwd` file. This was thought to be reasonably secure until dictionary attacks on encrypted passwords became common. At that point, the encrypted passwords, or "password hashes," were moved to the more secure `/etc/shadow` file. This new file also allowed password aging and expiration features to be implemented.

There are three pieces of information stored in a modern password hash:

\$1\$gCjLa2/Z\$6Pu0EK0AzfCjxjv2hoLOB/

1. **1**: The hashing algorithm. The number 1 indicates an MD5 hash. The number 6 appears when a SHA-512 hash is used.
2. **gCjLa2/Z**: The *salt* used to encrypt the hash. This is originally chosen at random. The salt and the unencrypted password are combined and encrypted to create the encrypted password hash. The use of a salt prevents two users with the same password from having identical entries in the **/etc/shadow** file.
3. **6Pu0EK0AzfCjxjv2hoLOB/**: The encrypted hash.

/etc/shadow format

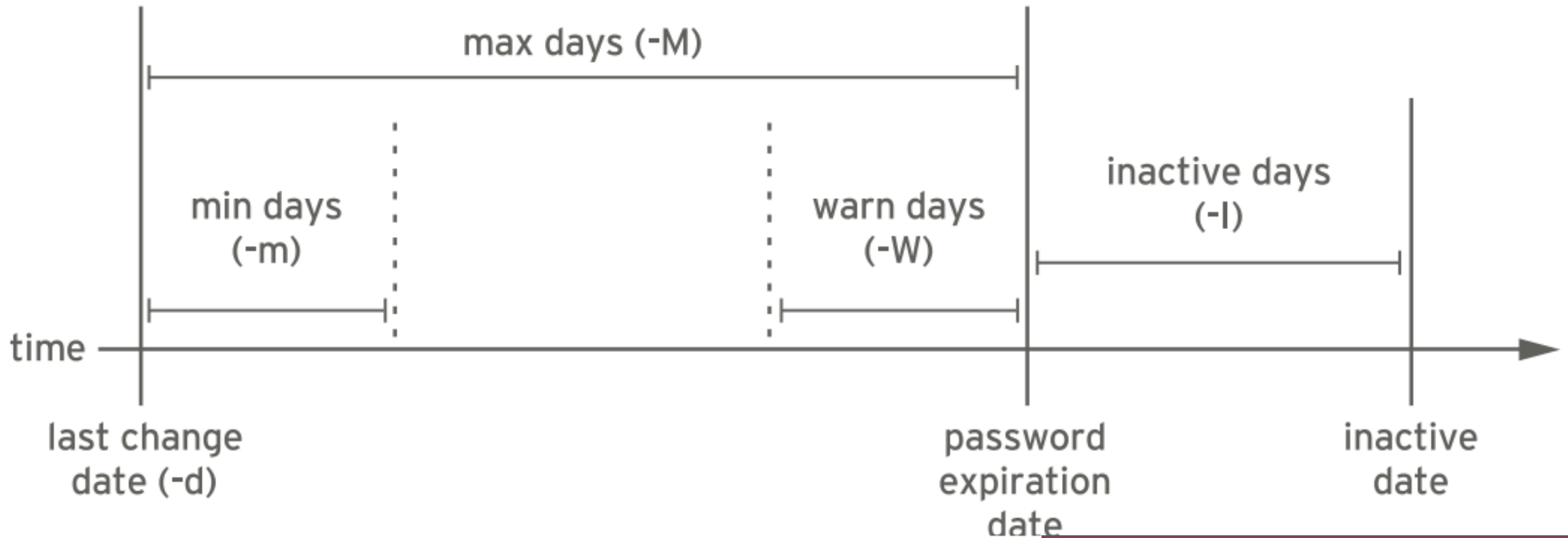
The format of **/etc/shadow** follows (nine colon-separated fields):

❶ name: ❷ password: ❸ lastchange: ❹ minage: ❺ maxage: ❻ warning: ❼ inactive: ❽ expire: ❾ blank

- ❶ The login *name*. This must be a valid account name on the system.
- ❷ The encrypted *password*. A password field which starts with an exclamation mark means that the password is locked.
- ❸ The date of the last *password change*, represented as the number of days since 1970.01.01.
- ❹ The *minimum* number of days before a password may be changed, where 0 means "no minimum age requirement."
- ❺ The *maximum* number of days before a password must be changed.
- ❻ The *warning* period that a password is about to expire. Represented in days, where 0 means "no warning given."
- ❼ The number of days an account remains active after a password has expired. A user may still log into the system and change the password during this period. After the specified number of days, the account is locked, becoming *inactive*.
- ❽ The account *expiration* date, represented as the number of days since 1970.01.01.
- ❾ This *blank* field is reserved for future use.

PASSWORD AGING

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



```
[root@serverX ~]#  
chage -m 0 -M 90 -W 7 -I 14 username
```

chage -d 0 *username* will force a password update on next login.

chage -l *username* will list a username's current settings.

chage -E *YYYY-MM-DD username* will expire an account on a specific day.

RESTRICTING ACCESS

With the **chage** command, an account expiration can be set. Once that date is reached, the user cannot log into the system interactively. The **usermod** command can "lock" an account with the **-L** option.

```
[student@serverX ~]$ sudo usermod -L elvis[student@serverX ~]$ su -  
elvisPassword: elvis  
su: Authentication failure
```

When a user has left the company, the administrator may lock and expire an account with a single **usermod** command. The date must be given as the number of days since 1970.01.01.

```
[student@serverX ~]$  
sudo usermod -L -e 1 elvis
```

The **nologin** shell

Sometimes a user needs an account with a password to authenticate to a system, but does not need an interactive shell on 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.

A common solution to this situation is to set the user's login shell to **/sbin/nologin**. If the user attempts to log into the system directly, the **nologin** "shell" will simply close the connection.

```
[root@serverX ~]# usermod -s /sbin/nologin student [root@serverX ~]# su - student
Last login: Tue Feb  4 18:40:30 EST 2014 on pts/0
This account is currently not available.
```



NOTE

Use of the **nologin** shell prevents interactive use of the system, but does not prevent all access. A user may still be able to authenticate and upload or retrieve files through applications such as web applications, file transfer programs, or mail readers.

End of Chapter 6

*Thank
you*

