Lecture- 7

Standard I/O & Text Processing

Scripting **Exploring the bash Shell**

.

Goal: Become familiar with the functions, syntax and use of several essential file and directory manipulation commands. Practice combining these commands together in useful ways to accomplish common user tasks.

System Setup: A working, installed red hat Enterprise Linux system with an unprivileged user account named student with a password of student.

Sequence 1: Directory and file Organization

Scenario: Once again files have managed to accumulate in your home directory and you have decided that it is time to organize things. This time you will use your knowledge of the bash shell to perform more complex file management tasks.

Deliverable: A more organized home directory, with files placed into the appropriate Subdirectories, and some files backed up to /tmp/archive.

Instructions:

1. Log in as user student with the password student. If you are the graphical environment, start a terminal by clicking Application->Accessories->terminal
2. Immediately after logging into the system, you should be in your home directory. Verify this with **pwd**.

[student@stationx ~] $ **Pwd**
/home/student

Note that you can also tell you are in your home directory by noting the ~ in your command prompt.

3. You will now use touch to create the files needed for this sequence. The details of how the expansion used in the following command works will be covered in a later Unit. For now, simply type the following line exactly as you see it (with the curly braces { } included, and an underscore character between the first few groups of sets). Have another nearby student or the instructor verifies the accuracy of your command before you press enter:

[student@stationx ~] **$ touch {report,memo,graph}_{Sep.Oct,nov,dec_{a,b,c_}_{1,2,3}**

4. Use the **ls** command to examine the results of the last command. You should find that it created 108 new, empty files (you do not need to count) in your home directory. These file represent the data files that you will use in the remainder of this sequence. If for some reason you do not see these files, ask the instructors for assistance; without these files, the remainder of this lab will not work.
5. In order to organize your files you must first create new directories. Use mkdir to create some subdirectories directly inside your home directory:

   [student@stationx ~] **$ mkdir a_reports**
   [student@stationx ~] **$ mkdir September October December**

   Again, use ls to examine your work
6. Create some additional subdirectories inside one of your new directories using the following commands.

   [student@stationx ~] **$ cd a_reports**
   To change to the directory. Then:
   [student@stationor,; a_reports] **$ mkdir one two three**

Use ls to verify that you have three new directories named one, two, and three under you're a_reports subdirectory.

7. Begin by moving all of the ''b'' reports out of your home directory and grouping them by month. When working with accomplished wildcard patterns. It is a good idea to pre-verify the operation to ensure you are operating on the correct files. One way to do this is to replace your command with a harmless command using the intended wildcard pattern.

   [student@stationx a_reports] **$ cd**
   [student@stationx ~] **$ ls -1 *dec_b_?**

Use should see the 9''December'', b" files listed. Move one of them to the December directory:

[student@stationX ~] **$ mv graph_dec_b_?december/**

Now move the reset of them with:

[student@stationX ~] **$ mv *dec_b_? December/**

List the contents of the December subdirectory to verify the move operation was successful:

[student@stationX ~] $ **ls -1 December**/

Total 9

-rw-rw-r--	1	student	student	0	Oct	16	22:17 graph_dec_b_1
-rw-rw-r--	1	student	student	0	Oct	16	22:17 graph_dec_b_2
-rw-rw-r--	1	student	student	0	Oct	16	22:17 graph_dec_b_3

```
-rw-rw-r--   1 student   student   0 Oct   16  22:17  memo_dec_b_1
-rw-rw-r--   1 student   student   0 Oct   16  22:17  memo_dec_b_2
-rw-rw-r--   1 student   student   0 Oct   16  22:17  memo_dec_b_3
-rw-rw-r--   1 student   student   0 Oct   16  22:17  report_dec_b_1
-rw-rw-r--   1 student   student   0 Oct   16  22:17  report_dec_b_2
-rw-rw-r--   1 student   student   0 Oct   16  22:17  report_dec_b_3
```

8. Move all of the remaining ‘’b’’ reports into their respective directories

[student@stationX ~] $ **mv *nov_b_? November/**
[student@stationX ~] $ **mv *Oct_b_? October/**
[student@stationX ~] **$ mv *sep_b_? September/**

9. You will now collect the ‘’a’’ reports into their respective corresponding numbered directories. Notice the use of ~ as shorthand for ‘’ your home directory’’ the combination of the wildcard and the pattern specifies all files that end in_a1 in your home directory.

[student@stationX ~] $ **cd a reports**
[student@stationX a_reports] **$ mv ~/*_a_1 one/**

The ‘’ September’’ al’’ files are old and no longer needed. Use ls to make sure you have created a pattern that matches only these file, then delete them, and verify that the other ‘al’ files were moved properly

[student@stationX a_reports] **$ cd one**
[student@stationX one] **$ ls *sep***
[student@stationX one] **$ rm *Sep***
[student@stationX one] **$ ls**
Report_dec_a_1 graph_oct_a_1 memo_no_a_1 report_dec_a_1
Report_oct_a_1 graph_nov_a_1 memo_dec_a_1 memo_oct_a_1
Report_nov_a_1

9. Move the final ‘’a_2’’ and ‘’a_3’’ reports into their respective directories. To make life interesting, we will move them from the current directory, using both relative and absolute pathnames. First, use pwd to identify the current directory.

[student@stationX one] $ pwd
/home/student/a_reports/one

Verify the pattern that references the ‘’a_2’’ files with ls, then move them using absolute pathnames:

[student@stationX one] $ **ls /home/student/*a_2***
[student@stationX one] $ **$ mv /home/student/*a_2/home/student/a_reports/two/**

Even though your current directory is /home/student/a_reports/one, you can move files from /home/student to /home/student/a_reports/two because you specified the files' using relative pathnames – in this case, absolute pathnames.

Now move the ''a_3'' files using relative pathnames. Again. First verify the pattern references the correct files.

[student@stationX one] $ **ls /../*a_3***
[student@stationX one] $ **mv ../../*a_3* ../three/**

10. Return to your home directory, and use ls to verify that the only files remaining in this directory are the ''c'' files (i.e.. graph_dec_c_1, graph_dec_c_2, …)

11. the ''c1'' and ''c2'' report files for each month are important, and you want to make backup of them in another directory:

[student@stationX ~] **$ mkdir /tmp/archive**
[student@stationX ~] **$ cp reports*{12} /tmp/archive/**

Additional, all the report files for the month of December should be backed up to the /tmp/archive directory. Note the use of the **–I** option to have **cp** prompt before overwriting any files.

[student@stationX ~] $ cp –I report_dec* /tmp/archive/
Cp: overwrite ./tmp/archive/report_dec_c_1,? n
Cp: overwrite ./tmp/archive/report_dec_c_2,? n

12. Now that you have backed up the few ''c'' files that are important to you, you want to delete all of the files still remaining in your home directory. Examination of the remaining files reveals that the wildcard *c* will match all of them. However, to ensure that you do not accidentally delete other files, try out the following commands, examining the output:

[student@stationX ~] **$ ls *c***
…output omitted…
[student@stationX ~] **$ ls -fd *c***
…output omitted…

13. Delete the remaining "c" files in your home directory. Once more we'll use ls before issuing a destructive command.

[student@stationX ~] $ ls *c*_[1 – 3]
…output omitted…

```
[student@stationX ~] $ rm *c*_[1 – 3]
[student@stationX ~] $ ls
A_reports  December  November  October  projects  September
```

Sequence 2: Automating tasks with shell scripts

Scenario: You would like to make it easier to automate the backup command you devised in the previous lab. The shell scripts you create will be built-upon in later labs

Deliverable: A script that prints information about and performs backups to a datesamped directory.

Instructions:

1. Consider the command you devised in a previous lab for creating a backup of /etc/sysconfig into a date_pecific directory. It should have looked something like:

   Cp_av/etc/sysconfig ~/backups/sysconfig-yyyymmdd

   This lab will have you create a simple shell script for replicating this command. Your script will have the advantage of determining the datestamp automatically. So that a destination does not have to be explicitly stated.

2. First you will need to devise a command that automatically generates a datestamp appropriate format. Consult the format section of man date and write the command you will need below. See the solutions sections for this exercise if you have trouble.

---

3. Now you are ready to create the script. Begin by logging in as root and creating a directory in your home directory called bin. this is the standard location for user's custom scripts.

```
[root@stationX ~] #mkdir ~/bin
```

4. Use a text editor, such as **nano** or **vi**, to create a new file called.

~ /bin/backup-sysconfig.sh. Because this file is a shell script, be sure to begin it with a ‘’ shbang” and a comment describing the script’s purpose:

 # ! /bin/bash
 # This script creates a backup of /etc/sysconfig
 # Into a datestanped subdirectory of ~ /backups/

1. Next, add a line that performs the **cp** command from your previous lab. Instead of typing an explicit date, use the $0 command substitution operator to run the date from the previous step each time backups-sysconfig.sh is executed.

Cp –av /etc/sysconfig ~backups/sysconfig-$ (date ‘+%y%m%d’_

2. Finally, add a line that prints some information about the backup that was just completed:

each “Backup of /etc/sysconfig completed at: $(date) “

7. Save the file. Your completed script should look like this:

 # ! /bin/bash
 # This script creates a backup of /etc/sysconfig
 # Into a datestamped subdirectory of ~ /backups/

Cp –av /etc/sysconfig ~ /backups/sysconfig-$ (date `+%y%m%d`)
Echo “backup of /etc/sysconfig completed at: $ (date)”

8. If you already have a backup with today’s datestamp. You should remove it before creating a new backup directory of the same name:

[root@stationX ~] # rm -rf ~ /backups/sysconfig-$(date `+%y%m%d`)

9. Before you can execute the script, you will need to make it executable. Run the following command to do this:

[**root@stationX ~] chmod u+x ~ /bin/backup-syscnfig.sh**

10. You are now be ready to try out your script; you should see something like the following:

[root@stationX ~] # **backup-sysconfig.sh**
` /etc/sysconfig` -> ` /home/brad/backups/sysconfig-200070129`
` /etc/sysconfig/irqbalance’ -> ` /home/brad/backups/sysconfig-200070129/grub’
…output truncated
Backup of /etc/sysconfig completed at: Mon Jan 18:13:22 EST 2007

If you have problems, double-check your script and try using **bash –x** in your shbang for diagnostic output

Tolls for Extracting Text:

- File contents: **less** and **cat**
- File Excerpts: **head** and **tall**
- Extract by column: out
- Extract by keyword: grep

Viewing File Contents Less and Cat:

- cat: dump one or more files to STDOUT
    - Multiple files are concatenated together
- Less: view file or STDIN one page at a time
    - Useful commands while viewing:
        - /text searches for test
        - n/w jumps to the next/previous match
        - V opens the file in text editor
    - less: is the pager used by man

Viewing File Excerpts head and tail:

- **head**: Display the first 10 lines of a file
    - Use –n to change number of lines displayed
- **tall**: Display the last 10 lines of a file
    - Use –n to change number of lines displayed
    - Use –f to change "follow" subsequent additions to the file
        - Very useful for monitoring log files!

The head command is used to display just the first few lines of a file. The default is 10 lines.

[student@stationX ~] **$ head /etc/passwd**
root:x:0:0:root:/root:/bin/bash
bin:x:1:bin:/bin:

daemon:x:2:2:daemon:sbin:
adm:x:3:4:adm:/var:adm:
ip:x:4:7:ip/var/spool/1pd:
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:
news:x:9:13:news:var/spool:news:

-n specifies the number of lines to display:

[student@stationx ~] $ head –n 3 /etc/passwd
Root:x:0:0:root:/root:/bin/bash
Bin:x:1:1:bin:/bin:
Daemon:x:2:2:daemon:/sbin

tail is used to display the last few lines of a file. The default is 10. **tail** is often used by the system administrator to read the most recent entries in log files.

[root@stationX ~] # tail –n 3 /var/log/cron
Root (10/13-18:01:00-658) cmd (run-parts /etc/cron.hourly)
CRON (10/15-13:50:00-781) STARTUP (fork ok)
Root (10/15-13:50:00-781 cmd (/sbin/rmmod-as)

Using **–f** causes **tail** to continue to display the file in "real time" showing additions to the end of the files as they occur. This is very useful for watching growing files. Such as the output of the make command. System administrator uses this feature to keep an eye on the system log using the following command:

[root@stationX ~] # **tail –f /var/log/messages**
Tail-f will continue to show updates to the file until Ctrl-c is pressed.

Extracting Text by Keyword grep:

- Prints lines of files or STDIN where a pattern is matched

$ grep 'john' /etc/passwd
$ date –help |grep year

- Use **–i** to search case-insensitively
- Use **–n** to print line numbers of matches
- Use –**Ax** to print include the x lines after each match
- Use **–Bx** to include the x lines before each match

Extracting Text by Column cut:

- Display specific columns of file or STDIN data

$ cut –d: –f1 /etc/passwd

$ grep root /etc/passwd | cut –d: -f7

- Use **–d** to specify the column delimiter (default is TAB)
- Use **–f** to specify the column to print
- Use **–c** to cut by characters

$ cut –c2 -5 /usr/share /dict /words

Tools for Analyzing Text:

- Text stats: **wc**
- Sorting Text: **sort**
- Comparing files: **diff** and **patch**
- Spell check: **aspell**

Gathering Text Statistics wc (word count):

- Counts words, lines, bytes and characters
- Can act upon a file or STDIN

$ wc story.txt

39 237 1901 story.txt

Use **–l** for only line count
Use **–w** for only word count
Use **–c** for only byte count
Use **–m** for character count (not displayed)

If you specify more than one file, we also reports totals for whatever values it has been told to count.

[student@stationX ~] $ wc .bash*
3 3 24 .bash_logout
13 29 191 .bash_profile
20 55 337 .bashrc
36 87 552 total

wc can also accept data on the standard input. This can be useful for counting the number of lines in a command's output. For example:

[student@stationX ~] $ ls /tmp ! wc -1

Shows us that **ls/tmp** produces 32 lines of output (even though by default your terminal will probably display them in columns). Therefore there are 32 files and directories in /tmp.

Sequence 1: Directory and file Organization

Scenario: Once again files have managed to accumulate in your home directory and you have decided that it is time to organize things. This time you will use your knowledge of the bash shell to perform more complex file management tasks.

Deliverable: A more organized home directory, with files placed into the appropriate Subdirectories, and some files backed up to /tmp/archive.

Instructions:

14. Log in as user student with the password student. If you are the graphical environment, start a terminal by clicking Application->Accessories->terminal

15. Immediately after logging into the system, you should be in your home directory. Verify this with **pwd**.

[student@stationx ~] $ **Pwd**
/home/student

Note that you can also tell you are in your home directory by noting the ~ in your command prompt.

16. You will now use touch to create the files needed for this sequence. The details of how the expansion used in the following command works will be covered in a later Unit. For now, simply type the following line exactly as you see it (with the curly braces { } included, and an underscore character between the first few groups of sets). Have another nearby student or the instructor verifies the accuracy of your command before you press enter:

[student@stationx ~] **$ touch {report,memo,graph}_{Sep.Oct,nov,dec_{a,b,c_}_{1,2,3}**

17. Use the **ls** command to examine the results of the last command. You should find that it created 108 new, empty files (you do not need to count) in your home directory. These file represent the data files that you will use in the remainder of this sequence. If for some reason you do not see these files, ask the instructors for assistance; without these files, the remainder of this lab will not work.

18. In order to organize your files you must first create new directories. Use mkdir to create some subdirectories directly inside your home directory:

[student@stationx ~] **$ mkdir a_reports**
[student@stationx ~] **$ mkdir September October December**

Again, use ls to examine your work

19. Create some additional subdirectories inside one of your new directories using the following commands.

[student@stationx ~] **$ cd a_reports**
To change to the directory. Then:
[student@stationor,; a_reports] **$ mkdir one two three**

Use ls to verify that you have three new directories named one, two, and three under you're a_reports subdirectory.

20. Begin by moving all of the ‘’b’’ reports out of your home directory and grouping them by month. When working with accomplished wildcard patterns. It is a good idea to pre-verify the operation to ensure you are operating on the correct files. One way to do this is to replace your command with a harmless command using the intended wildcard pattern.

[student@stationx a_reports] **$ cd**
[student@stationx ~] **$ ls -1 *dec_b_?**

Use should see the 9’’December’’, b” files listed. Move one of them to the December directory:

[student@stationX ~] **$ mv graph_dec_b_?december/**

Now move the reset of them with:

[student@stationX ~] **$ mv *dec_b_? December/**

List the contents of the December subdirectory to verify the move operation was successful:

[student@stationX ~] $ **ls -1 December**/

Total 9

-rw-rw-r--	1	student	student	0	Oct	16	22:17 graph_dec_b_1
-rw-rw-r--	1	student	student	0	Oct	16	22:17 graph_dec_b_2
-rw-rw-r--	1	student	student	0	Oct	16	22:17 graph_dec_b_3
-rw-rw-r--	1	student	student	0	Oct	16	22:17 memo_dec_b_1
-rw-rw-r--	1	student	student	0	Oct	16	22:17 memo_dec_b_2
-rw-rw-r--	1	student	student	0	Oct	16	22:17 memo_dec_b_3
-rw-rw-r--	1	student	student	0	Oct	16	22:17 report_dec_b_1
-rw-rw-r--	1	student	student	0	Oct	16	22:17 report_dec_b_2
-rw-rw-r--	1	student	student	0	Oct	16	22:17 report_dec_b_3

21. Move all of the remaining ‘’b’’ reports into their respective directories

[student@stationX ~] $ **mv *nov_b_? November/**
[student@stationX ~] $ **mv *Oct_b_? October/**
[student@stationX ~] **$ mv *sep_b_? September/**

9. You will now collect the ‘’a’’ reports into their respective corresponding numbered directories. Notice the use of ~ as shorthand for ‘’ your home directory’’ the combination of the wildcard and the pattern specifies all files that end in_a1 in your home directory.

[student@stationX ~] $ **cd a reports**

[student@stationX a_reports] **$ mv ~/*_a_1 one/**

The ‘’ September’’ al’’ files are old and no longer needed. Use ls to make sure you have created a pattern that matches only these file, then delete them, and verify that the other ‘al’ files were moved properly

[student@stationX a_reports] **$ cd one**
[student@stationX one] **$ ls *sep***
[student@stationX one] **$ rm *Sep***
[student@stationX one] **$ ls**
Report_dec_a_1 graph_oct_a_1 memo_no_a_1 report_dec_a_1
Report_oct_a_1 graph_nov_a_1 memo_dec_a_1 memo_oct_a_1
Report_nov_a_1

22. Move the final ‘’a_2’’ and ‘’a_3’’ reports into their respective directories. To make life interesting, we will move them from the current directory, using both relative and absolute pathnames. First, use pwd to identify the current directory.

[student@stationX one] $ pwd
/home/student/a_reports/one

Verify the pattern that references the ‘’a_2’’ files with ls, then move them using absolute pathnames:

[student@stationX one] $ **ls /home/student/*a_2***
[student@stationX one] $ **$ mv /home/student/*a_2/home/student/a_reports/two/**

Even though your current directory is /home/student/a_reports/one, you can move files from /home/student to /home/student/a_reports/two because you specified the files’ using relative pathnames – in this case, absolute pathnames.

Now move the ‘’a_3’’ files using relative pathnames. Again. First verify the pattern references the correct files.

[student@stationX one] $ **ls /../*a_3***
[student@stationX one] $ **mv ../../*a_3* ../three/**

23. Return to your home directory, and use ls to verify that the only files remaining in this directory are the ‘’c’’ files (i.e.. graph_dec_c_1, graph_dec_c_2, …)

24. the ‘’c1’’ and ‘’c2’’ report files for each month are important, and you want to make backup of them in another directory:

[student@stationX ~] **$ mkdir /tmp/archive**
[student@stationX ~] **$ cp reports*{12} /tmp/archive/**

Additional, all the report files for the month of December should be backed up to the /tmp/archive directory. Note the use of the **–I** option to have **cp** prompt before overwriting any files.

```
[student@stationX ~] $ cp –I report_dec* /tmp/archive/
Cp: overwrite ./tmp/archive/report_dec_c_1,? n
Cp: overwrite ./tmp/archive/report_dec_c_2,? n
```

25. Now that you have backed up the few ‘’c’’ files that are important to you, you want to delete all of the files still remaining in your home directory. Examination of the remaining files reveals that the wildcard *c* will match all of them. However, to ensure that you do not accidentally delete other files, try out the following commands, examining the output:

```
[student@stationX ~] $ ls *c*
…output omitted…
[student@stationX ~] $ ls -fd *c*
…output omitted…
```

26. Delete the remaining “c” files in your home directory. Once more we’ll use ls before issuing a destructive command.

```
[student@stationX ~] $ ls *c*_[1 – 3]
…output omitted…
[student@stationX ~] $ rm *c*_[1 – 3]
[student@stationX ~] $ ls
A_reports  December  November  October  projects  September
```

Sequence 2: Automating tasks with shell scripts

Scenario: You would like to make it easier to automate the backup command you devised in the previous lab. The shell scripts you create will be built-upon in later labs

Deliverable: A script that prints information about and performs backups to a datesamped directory.

Instructions:

3. Consider the command you devised in a previous lab for creating a backup of /etc/sysconfig into a date_pecific directory. It should have looked something like:

   Cp_av/etc/sysconfig ~/backups/sysconfig-yyyymmdd

   This lab will have you create a simple shell script for replicating this command. Your script will have the advantage of determining the datestamp automatically. So that a destination does not have to be explicitly stated.

4. First you will need to devise a command that automatically generates a datestamp appropriate format. Consult the format section of man date and write the command you will need below. See the solutions sections for this exercise if you have trouble.

---

3. Now you are ready to create the script. Begin by logging in as root and creating a directory in your home directory called bin. this is the standard location for user's custom scripts.

[root@stationX ~] #mkdir ~/bin

4. Use a text editor, such as **nano** or **vi**, to create a new file called.
~ /bin/backup-sysconfig.sh. Because this file is a shell script, be sure to begin it with a '' shbang" and a comment describing the script's purpose:

 # ! /bin/bash
 # This script creates a backup of /etc/sysconfig
 # Into a datestanped subdirectory of ~ /backups/

3. Next, add a line that performs the **cp** command from your previous lab. Instead of typing an explicit date, use the $0 command substitution operator to run the date from the previous step each time backups-sysconfig.sh is executed.

Cp –av /etc/sysconfig ~backups/sysconfig-$ (date '+%y%m%d'_

4. Finally, add a line that prints some information about the backup that was just completed:

each "Backup of /etc/sysconfig completed at: $(date) "

7. Save the file. Your completed script should look like this:

 # ! /bin/bash
 # This script creates a backup of /etc/sysconfig
 # Into a datestamped subdirectory of ~ /backups/

Cp –av /etc/sysconfig ~ /backups/sysconfig-$ (date `+%y%m%d`)
Echo "backup of /etc/sysconfig completed at: $ (date)"

8. If you already have a backup with today's datestamp. You should remove it before creating a new backup directory of the same name:

[root@stationX ~] # rm -rf ~ /backups/sysconfig-$(date `+%y%m%d`)

9. Before you can execute the script, you will need to make it executable. Run the following command to do this:

[**root@stationX ~] chmod u+x ~ /bin/backup-syscnfig.sh**

10. You are now be ready to try out your script; you should see something like the following:

[root@stationX ~] # **backup-sysconfig.sh**
` /etc/sysconfig` -> ` /home/brad/backups/sysconfig-200070129`
` /etc/sysconfig/irqbalance' -> ` /home/brad/backups/sysconfig-200070129/grub'
…output truncated
Backup of /etc/sysconfig completed at: Mon Jan 18:13:22 EST 2007

If you have problems, double-check your script and try using **bash –x** in your shbang for diagnostic output

Tolls for Extracting Text:

- File contents: **less** and **cat**
- File Excerpts: **head** and **tall**
- Extract by column: out
- Extract by keyword: grep

Viewing File Contents Less and Cat:

- cat: dump one or more files to STDOUT
    - Multiple files are concatenated together
- Less: view file or STDIN one page at a time
    - Useful commands while viewing:
        - /text searches for test
        - n/w jumps to the next/previous match
        - V opens the file in text editor
    - less: is the pager used by man

Viewing File Excerpts head and tail:

- **head**: Display the first 10 lines of a file
    - Use –n to change number of lines displayed
- **tall**: Display the last 10 lines of a file
    - Use –n to change number of lines displayed
    - Use –f to change "follow" subsequent additions to the file
        - Very useful for monitoring log files!

The head command is used to display just the first few lines of a file. The default is 10 lines.

[student@stationX ~] **$ head /etc/passwd**
root:x:0:0:root:/root:/bin/bash
bin:x:1:bin:/bin:
daemon:x:2:2:daemon:sbin:
adm:x:3:4:adm:/var:adm:
ip:x:4:7:ip/var/spool/1pd:
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:
news:x:9:13:news:var/spool:news:

-n specifies the number of lines to display:

[student@stationx ~] $ head –n 3 /etc/passwd
Root:x:0:0:root:/root:/bin/bash
Bin:x:1:1:bin:/bin:
Daemon:x:2:2:daemon:/sbin

tail is used to display the last few lines of a file. The default is 10. **tail** is often used by the system administrator to read the most recent entries in log files.

[root@stationX ~] # tail –n 3 /var/log/cron
Root (10/13-18:01:00-658) cmd (run-parts /etc/cron.hourly)

CRON (10/15-13:50:00-781) STARTUP (fork ok)
Root (10/15-13:50:00-781 cmd (/sbin/rmmod-as)

Using **–f** causes **tail** to continue to display the file in "real time" showing additions to the end of the files as they occur. This is very useful for watching growing files. Such as the output of the make command. System administrator uses this feature to keep an eye on the system log using the following command:

[root@stationX ~] # **tail –f /var/log/messages**
Tail-f will continue to show updates to the file until Ctrl-c is pressed.

Extracting Text by Keyword grep:

- Prints lines of files or STDIN where a pattern is matched

$ grep 'john' /etc/passwd
$ date –help |grep year

- Use **–i** to search case-insensitively
- Use **–n** to print line numbers of matches
- Use –**Ax** to print include the x lines after each match
- Use **–Bx** to include the x lines before each match

Extracting Text by Column cut:

- Display specific columns of file or STDIN data

$ cut –d: –f1 /etc/passwd
$ grep root /etc/passwd | cut –d: -f7

- Use **–d** to specify the column delimiter (default is TAB)
- Use **–f** to specify the column to print
- Use **–c** to cut by characters

$ cut –c2 -5 /usr/share /dict /words

Tools for Analyzing Text:

- Text stats: **wc**
- Sorting Text: **sort**
- Comparing files: **diff** and **patch**
- Spell check: **aspell**

Gathering Text Statistics wc (word count):

- Counts words, lines, bytes and characters
- Can act upon a file or STDIN

$ wc story.txt
39 237 1901 story.txt

Use **–l** for only line count
Use **–w** for only word count
Use **–c** for only byte count
Use **–m** for character count (not displayed)

If you specify more than one file, we also reports totals for whatever values it has been told to count.

```
[student@stationX ~] $ wc .bash*
3       3       24      .bash_logout
13      29      191     .bash_profile
20      55       337    .bashrc
36      87       552     total
```

wc can also accept data on the standard input. This can be useful for counting the number of lines in a command's output. For example:

```
[student@stationX ~] $ ls /tmp ! wc -1
```

Shows us that **ls/tmp** produces 32 lines of output (even though by default your terminal will probably display them in columns). Therefore there are 32 files and directories in /tmp.