

Course 144 Supplementary Materials

UNIX Fundamentals



Background to UNIX Command Fundamentals

- **This appendix provides a overview of critical commands and concepts**
 - Prerequisite knowledge attendees should already have
 - Some (but not all!) of the most important commands for users
- **Attendees of this course should already know the commands and concepts discussed on all pages in this appendix, except for the final three titled “System Administration”**
 - Either Course 143 or Course 428 would provide the needed background
- **As a UNIX/Linux system administrator, it is your responsibility to first be a skilled UNIX/Linux user**
 - Use the `man` command to learn about commands
 - List of commands by topic: `man -k topicname`
 - Details on one command: `man commandname`
 - UNIX documentation uses italics to indicate variable parts of examples, as in the above
 - Polish your skill by using the commands!



143
428

Editing Text Files

Almost all UNIX/Linux configuration is done with text files. This means that you must be able to accurately and efficiently modify these files with a standard UNIX/Linux text editor. The only interactive editor you can count on having available is the `vi` editor.

Newer versions of `vi`, actually `vim` (for “VI Improved”), may support use of the special keys on the keypad to the right of the main keyboard: the arrow keys, `<Page Up>`, `<Page Down>`, `<Delete>`, and `<Insert>`. *However, those keys will not function at all times!*

If you are going to be a system administrator, you may be called upon to edit text files when those keys do not function. You must be able to use the standard `vi` commands for cursor motion (`h`, `j`, `k`, `l`, `w`, `b`, `^D`, `^U`, etc.), entry into and exit from insertion mode (`i`, `a`, `o`, `O`, `Escape`, etc.), and deletion (`x`, `dw`, `dd`, etc.).

Filename Wildcards

- *** Matches any string, including an empty string
"foo*" matches "foo", "foot", and "football"
- ?** Matches any one character
"foo?" matches "food" and "foot", but not "football"
- [. . . .]** Matches any single character from the list

"foo[ldt]"	Matches any one of "food", "fool", or "foot"
"foo[a-c]"	Matches any one of "fooa", "foob", or "fooc"
"foo[a-cx-z]"	Matches any one of "fooa", "foob", "fooc", "foox", "fooy", or "fooz"
"[a-cmrx-z]"	Matches any one of "a", "b", "c", "m", "n", "x", "y", or "z"

Be careful!

"foo[d,t]ball" Matches any one of "foodball", "foo,ball", or "football!"

Shell Fundamentals: I/O

myprog

Run the program `myprog` and allow standard output and standard error to print on the terminal screen

myprog > saveit

Send standard output to a file named `saveit`. This overwrites `saveit` if it exists!

myprog >> saveit

Append standard output to `saveit`. Any data in `saveit` is preserved, and the new output is added to the end.

myprog > saveit 2> oops

Send standard output to `saveit` and standard error (I/O stream #2) to `oops`

myprog > saveit 2>&1

Send standard output to `saveit`, and also send standard error there

myprog | otherprog

Build a pipeline—the output of `myprog` is used as the input of `otherprog`

Shell Fundamentals: Running Programs

You do not need to be in any particular directory to run a program, so long as:

- 1: The program is in a directory listed in your PATH environment variable, or
- 2: You specify the full path to the program

Reasonable PATH for users includes these directories:

```
/bin, /usr/bin, /usr/X11R6/bin, /usr/local/bin
```

Reasonable PATH for `root` adds these directories:

```
/sbin, /usr/sbin, /usr/local/sbin
```

If the `/usr/local` hierarchy is not carefully controlled, leave `/usr/local/*bin` out of PATH for `root`!

Any I/O redirection is relative to your current working directory

Scenario: You have data files in the directory `/var/log/something`. You want to process them with the program `/usr/local/sbin/analyzer`, which uses its command-line arguments as sources of input. Finally, you want to save the output in a file named `analysis` in the current directory. Just do this:

```
/usr/local/sbin/analyzer /var/log/something/* > analysis
```

Listing Files

```
ls      List the names of the files (remember, directories are just a special type of file)
ls -a   List all the files, even those with names starting with "."
ls -l   Long listing of the files and their attributes
ls -d   List just the directory itself, not its contents
```

Combine these to do something like: *List the attributes of just those files (but not their contents if they are directories) with names starting with capital letters:*

```
ls -ld [A-Z]*
```

Reading the output of `ls -l`:

```
drwxr-xr-x 3 juser bigproject 1024 Aug 13 08:14 Data
-rw-r--r-- 1 juser bigproject 16192 Aug 22 20:41 README
```

		Owner	Group	Size in bytes	Modification timestamp	Filename	

<p>File type</p> <ul style="list-style-type: none"> d Directory l Symbolic link b Block device c Character device - Regular file 	<p>Permissions:</p> <ul style="list-style-type: none"> First 3 for owner (juser in this example) Second 3 for members of the group (bigproject in this example) Last 3 for everyone else r = read, w = write, x = execute (or search in the case of directories) s = set-user-ID or set-group-ID t = sticky bit (you cannot remove other user's files from this directory)
--	---

Manipulating Files and Directories

<code>cp foo bar</code>	Copy file <code>foo</code> to <code>bar</code> . If <code>bar</code> is a directory, create the file <code>bar/foo</code> . Otherwise, <code>bar</code> will be a file that is a copy of <code>foo</code> .
<code>mv foo bar</code>	Move file <code>foo</code> to <code>bar</code> . If <code>bar</code> is a directory, create the file <code>bar/foo</code> . Otherwise, <code>foo</code> will be renamed <code>bar</code> .
<code>mkdir foo</code>	Create a directory named <code>foo</code>
<code>rmdir foo</code>	Remove a directory named <code>foo</code>
<code>touch foo</code>	If <code>foo</code> exists, change its modification timestamp . If it does not exist, create it as an empty file.
<code>ln -s foo bar</code>	Create a symbolic link named <code>foo</code> pointing to <code>bar</code>
<code>rm foo</code>	Remove <code>foo</code>
<code>rm -r foo</code>	Recursively remove the directory <code>foo</code> and its contents
<code>rm -rf foo</code>	Recursively remove the directory <code>foo</code> and its contents and force it to happen. (Do not bother me with warning messages!)

Examining File Contents

<code>more foo</code>	View the contents of <code>foo</code> one screen at a time. Press <code><Spacebar></code> to move forward one screen, <code></code> to move back one screen, and <code><q></code> to quit.
<code>cat foo</code>	Print all the contents of <code>foo</code> to the screen
<code>cmp foo bar</code>	Compare —answer the question, “Are <code>foo</code> and <code>bar</code> identical?”
<code>diff foo bar</code>	Show the differences between <code>foo</code> and <code>bar</code>
<code>grep blah foo</code>	Print just those lines of <code>foo</code> containing the literal string <code>blah</code>
<code>grep -i blah foo</code>	...except ignore the case of the letters in <code>blah</code>
<code>grep -w blah foo</code>	...only when <code>blah</code> appears as an isolated word
<code>grep -iw blah foo</code>	...as an isolated word and ignoring case
<code>grep -v blah foo</code>	...only the lines that do not contain the string <code>blah</code>

Dealing With Lots of Data

```
myprog | more    Run the program myprog and display its standard output one screen
                  at a time with more

myprog | less    The program less is like more except with more features—less
                  lets you back up a page at a time with <b>

myprog | grep -i blah | less
                  Run the program myprog, displaying only the lines containing the
                  string blah (in any case), displaying the result one page at a time
```

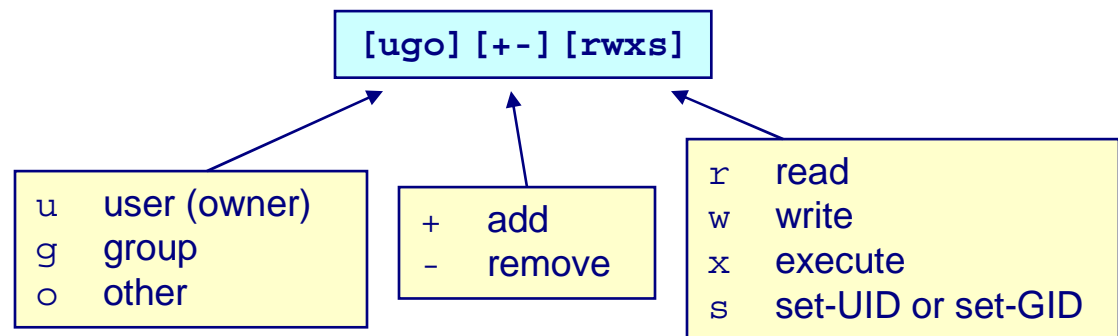
Changing File Permissions

<code>chmod 700 mydata</code>	Change the permissions of the file <code>mydata</code> to mode 700
<code>chmod o-w mydata</code>	Remove the permission for others to write the file <code>mydata</code> , but leave the other permissions alone
<code>chmod -R o-w mydata</code>	Apply that change recursively to the directory <code>mydata</code> and its contents

Permissions can be specified in octal.
Use one of these bit patterns for each of user, group and other, in that order:

7	<code>rwX</code>
6	<code>rw-</code>
5	<code>r-X</code>
4	<code>r--</code>
1	<code>--X</code>
0	<code>---</code>

Permissions can be specified symbolically:



Who Am I and Where Am I?

<code>id</code>	List your credentials—user ID, primary group ID, other groups
<code>pwd</code>	Print the current working directory
<code>hostname</code>	Print the computer's host name
<code>ifconfig -a</code>	Display the computer's IP address(es)

Controlling Processes

<code>ps axuw</code>	List many details about all running processes. On Solaris, you must provide the full path: <code>/usr/ucb/ps axuw</code>
<code>kill -HUP 123</code>	Send a <code>HUP</code> signal to the process with PID 123. Many daemons interpret a <code>HUP</code> signal to mean “Keep running, but re-read your configuration file.”
<code>kill -TERM 123</code>	Send a <code>TERM</code> signal to the process with PID 123. A well-written process is given a chance to shut down cleanly.
<code>kill -KILL 123</code>	Send a <code>KILL</code> signal to the process with PID 123. Does not allow a clean shutdown like <code>TERM</code> , but sometimes a blunt weapon is needed!
<code>pkill -HUP xinetd</code>	Like <code>kill</code> , except you can specify processes by program name, not PID

Shut UNIX down cleanly and halt

<code>init 0</code>	Solaris and Linux
<code>halt</code>	BSD

Shut UNIX down cleanly and reboot

<code>init 6</code>	Solaris and Linux
<code>reboot</code>	BSD

Go to single-user mode (maintenance mode, root on the console)

<code>init 1</code>	Solaris and Linux
<code>kill -s TERM 1</code>	BSD

Go to full multi-user mode

<code>init 3</code>	Solaris
<code>init 3</code>	Linux (for a text-only environment)
<code>init 5</code>	Linux (for a graphical environment)
<code>^D</code>	On BSD, exit from the single-user mode shell

System Administration: Managing Users and Groups

<code>useradd -m fred</code>	Create a user named <code>fred</code> and create the home directory <code>/home/fred</code> based on the contents of <code>/etc/skel</code>
<code>userdel fred</code>	Delete the user <code>fred</code> (home directory and all files remain)
<code>passwd fred</code>	Assign a password to the user <code>fred</code>
<code>groupadd -g 100 wheel</code>	Create a group named <code>wheel</code> with group ID 100

User accounts are defined in:

<code>/etc/passwd</code>	Login, UID, primary GID, real name (“GECOS field”), home directory, shell
<code>/etc/shadow</code>	Login, hashed password, account and password aging control

Groups are defined in `/etc/group`: Group name, GID, list of members

System Administration: Changing File Owner and Group

<code>chown fred mydata</code>	Change the owner of the file <code>mydata</code> to the user <code>fred</code>
<code>chgrp sys mydata</code>	Change the group of the file <code>mydata</code> to the group <code>sys</code>
<code>chown root:sys mydata</code>	Change the owner of the file <code>mydata</code> to the user <code>root</code> , and change the group to <code>sys</code>
<code>chown -R root:sys foo</code>	Apply that user:group change recursively to the directory <code>foo</code> and its contents