Course 433 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

143
428

- **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!

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-cmnx-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**

File type
`d` Directory
`l` Symbolic link
`b` Block device
`c` Character device
`-` Regular file

Permissions:
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

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

Examining File Contents

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

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

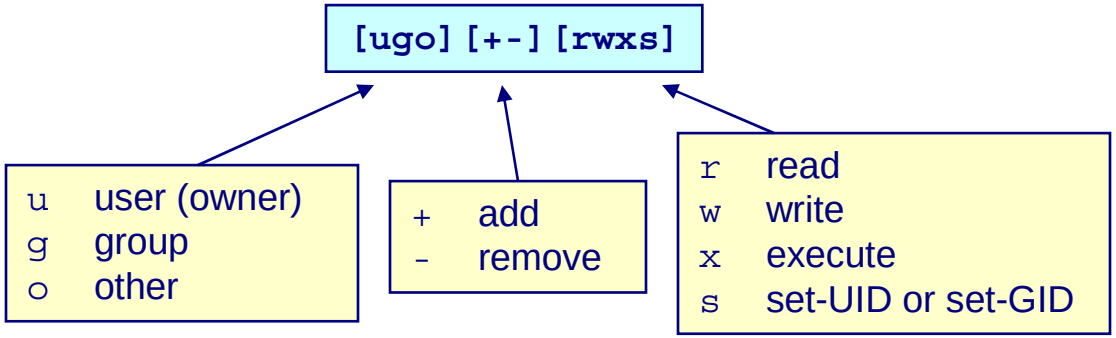
Changing File Permissions

`chmod 700 mydata`	Change the permissions of the file `mydata` to mode 700
`chmod o-w mydata`	Remove the permission for others to write the file `mydata`, but leave the other permissions alone
`chmod -R o-w mydata`	Apply that change recursively to the directory `mydata` 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	rwx
6	rw-
5	r-x
4	r--
1	--x
0	---

Permissions can be specified symbolically:



Who Am I and Where Am I?

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

Controlling Processes

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

System Administration: Controlling System State

Shut UNIX down cleanly and halt

`init 0`	Solaris and Linux
`halt`	BSD

Shut UNIX down cleanly and reboot

`init 6`	Solaris and Linux
`reboot`	BSD

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

`init 1`	Solaris and Linux
`kill -s TERM 1`	BSD

Go to full multi-user mode

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

System Administration: Managing Users and Groups

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

User accounts are defined in:

`/etc/passwd`	Login, UID, primary GID, real name ("GECOS field"), home directory, shell
`/etc/shadow`	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

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