File Management Commands

Cut →

1. Allows you to strip text out of file and display the cut text on the screen or redirect to a file

2. Most often used to cut fields within a file.

3. Command does not alter the original file and automatically opens the file for you.

cut options filename.

-d : option used to identify the delimiters in the file.

- f : option used to identify which field you want.

Example: 1001 : USA : Washington DC

ID — Country — Capital

cut -d: -f1 file.txt] displays the first field 1001

cut -d: -f1,3 file.txt] display the 1st and 3rd field.

paste → 1. Merges data from one file to another.

paste file1 file2

Merges each line from file2 with each line of file1.

Example, last.dat has

L1: Smith
L2: Jones
L3: Adam

] Last names

first.dat has

L1: Joe
L2: Mary
L3: Sue

] First names

paste first.dat last.dat, produces

Joe Smith
Mary Jones
Sue Adams

Uniq → 1. Used to find duplicate lines from a sorted file.

2. Used to find redundant records in a file.

Uniq filename.

-c option

uniq -c file

→ Counts the number of occurrences of a line in a file.

— Helpful for identifying the number of duplicates in a file.

Viewing File Contents

File Statistics

stat filename ← Provides a complete rundown of the status of a file on the filesystem

```
$ stat test10
  File: "test10"
  Size: 6        Blocks: 8        Regular File
  Device: ____   Uid: _           Inode   Links:
  Acces: Time
  Modify: Time
  Chang: Time
```

Viewing File Type

1. stat does not tell you the type of the file.
2. Good to know if you want to display the contents of file on screen e.g. cat binary-file would produce gibberish.

file filename

```
$ file test1
test1: Ascii text
$ file myscript
myscript: Bourne shell script text executable
```

file command classifies file into

1. Text file : contains printable characters
2. Executable : Files that run on the system
3. Data file : File containing nonprintable binary characters,

Viewing Parts of a File

1. If the information you are looking for is at the top or buried at the bottom of a large file, then using cat or more is cumbersome.

tail → - Displays the last group of lines in a file.
- Default shows last 10 lines.

tail -n 20 filename
(20 lines from the bottom.)

head -n# file — shows top n lines

tail -f filename

Allows you to peek inside a file as it is being used by other processes.
tail command stays active and continues to display new lines as they appear in the text file.
Great way to monitor log files.

Sort → Sorts the data lines in a text file using standard sorting rules.

```
$ cat file1
one
two
three
four
five
$ sort file1
five
four
one
three
two
```

alphabet order

When sorting numbers, sort by default interprets numbers as characters.

Used -n

Example,

```
$ cat file2
1
2
100
45
3
10
145
75
$ sort file2
1
10
100
145
2
3
45
75
```

Used $

sort -n file2 for correct sorting

If there are dates in the text file use -M option to sort by month of the year

-k and -t options useful when sorting data that uses fields.

field separator: -t ';'
field #: -k 3
numeric sorting: -n

```
$ sort -t ';' -k 3 -n filename
```

sort a file with a file separator (;), using the 3rd field and do the sorting in numeric order.

Example, sorting the process list in the assignment.

Compressing Data

1. Good for compressing large files
2. Good for archiving
3. Zip very popular among Windows users.

Linux Compression Utilites

bzip2	.bz2	← Burrows-Wheeler block sorting text compression
compress	.Z	← Original Unix file compression utility; obscure
gzip	.gz	← GNU compression utility.
zip	.zip	← Unix version of the PKZIP program for windows.

gzip

gzip → compress files

gzcat → display contents of compressed text files

gunzip → uncompressing files.

Usage gzip filename

zip

zip → Creates a compressed file containing files and directories

unzip → extract files and directories from a compressed .zip file

Usage zip -r zipfilename directory

↑ ↳ Ideal for archiving entire directory structure.

Recurses through the directory and adds each file and directory found to a zip file~~s to add~~.

Archiving Data using tar

- Zip works great, but it is not the standard utility used in Linux and Unix worlds.

- tar command originally used to write files to a tape device for archiving.

tar function [options] object1 object2 ....

function options:
- -A append an existing tar archive file to another existing tar archive file.
- -c create a new tar archive
- -r append files to a tar archive
- -x extract files.

Command Options:
- -f output results to file
- -v list files as they are processed

Example: tar -cvf test.tar test/ test1/

Creates an archive file called test.tar containing both the directories test and test1.

tar -tf test.tar ← lists (but not extracts) contents of the archive

tar -xvf test.tar ← extracts.

Decoding File Permissions

- rwx rwx r-x

Owner group others.

	Binary	Octal
rwx	111	$2^2+2^1+2^0=7$
---	000	0
r--	100	$2^2+0+0=4$

Default file permissions

- How do the file permission come when creating a new file?

Say using touch newfile

umask → shows the default permissions.

```
$ umask
0022
```

Octal mode

Sticky bit
Security feature

The umask value is subtracted from the full permission set for an object.

Default permissions for the user

For a File : 666 (rw- rw- rw-)
For a directory : 777 (rwx rwx rwx)

So, 666 − 022 = 644 and 777 − 022 = 755

Working with Editors

— Emacs
— Gedit (graphical)

vim : — Vim editor works with data in a memory buffer.

— If you start vim without a filename, or if file doesn't exist, vim opens a new buffer area for editing.

Two modes → Normal mode } opens in normal mode
→ Insert mode. (press i)

vim inserts every key you type in the buffer

— Lots of nifty features such as search and substitute using regular expressions.

— Copying deleting multiple lines at the same time

— Requires remembering lots of commands.

- :q → quit if no changes made to buffer data.
- :q! → quit and discard changes.
- :wq → save the buffer data to the file and quit.

Creating a Script File

1) Specify the shell using in the first line of the file.

```
#!/bin/bash
```

Tells the shell which shell to run the script under.

You can be running a bash shell, but run the script in tcsh.

← normally a comment in shell and not interpreted.

However, the first line is special.

Save as test1

```
#!/bin/bash
# This script displays date and who's logged
date
who
```

each command on a new line

or `date; who`

You can put multiple commands on same line separated by semi-colon (;)

2) Running script

```
$ test1
bash: test1: command not found
```

Problem is shell can not find your script Location Not listed in PATH

You can do two things:

1) Add directory where shell scripts located to PATH variable

PATH=$PATH: path of the directory

2) Use an absolute or relative file path to reference the shell script in the prompt

{ Use single dot operator to reference to current directory }

```
$ ./test1
bash: ./test1: Permission denied
```

Shell found the script, you do not have permision to execute the file.

make it executable

```
$chmod u+x test1.
$ ./test1
```

Succes!

Variables

- Used to represent data
- Composed of the variable name and the data.
- To refer to the contents of a variable, use the $ infront of the variable name.

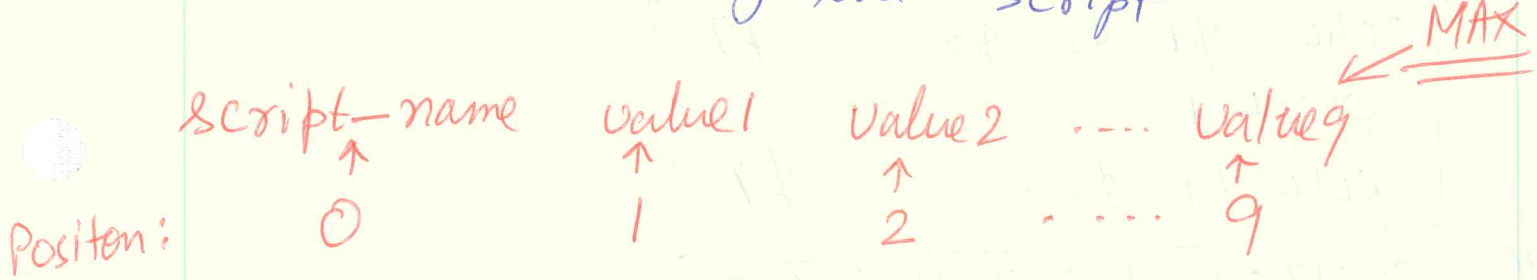
```
x=5 . echo $x
```

Explicit variable declaration ← not required

```
declare -i x      ← integer variable x
declare -i -r x   ← integer read only variable x.
```

Understanding Positional Parameters

- Refer to the numeric position of the value in the list that follows the script name
- Very useful for providing user input when executing the script



— To refer to the contents of a parameter, you precede the parameter with a $ followed by position number (e.g. $1, $2 ... $9)

$0 → refers to the script.

(cannot be used for passing values to the script)

Example Script called · display-it

```
echo $1 $2
```

run display-it a b

Output will be a b

What happens when you have more than 9 positional parameters?

shift n ← number of positional parameters you want to shift.

ShiftThem

```
echo $1 $2 $3 ... $9
Shift 1
echo $1 $2 ... $9
```

ShiftThem 1 2 3 4 5 6 7 8 9 10

Output:

1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9 10

Shifting more than 1 number

Imagine you need to add together 10 donations raised at an auction. You can only pass 9 values so you need to shift the values.

```
Donated Sum
((Donation = $1+$2+$3+$4+$5+$6+$7+$8+$9))
  Shift 9
  ((Donation = $Donation + $1))
   echo $Donation
```

DonatedSum 100 200 300 5 6 7 9 110 125

Special Parameters

? → exit status of a command.

→ number of positional parameters.

```
echo "The number of parameters is" $#
```

↑ Very useful for Usage Clause.

Exit codes

max 255 modulo256 remainder when divided by 256

0 – Success
1 – Unknown error (fail)
126 – Misuse of shell
127 – command not found
255 – exit status out of range

echo $?

17

- You can use your own exit codes in script
- You can use variables in exit command parameters.

Example,

```
#!/bin/bash
var1=10
var2=30
((var3=var1+var2))
exit $var3
```

we get exit code 40

- If you use exit code greater than 255, then you get the mod 256 value as the exit code. Example, exit code 300 would result in 44 being the displayed exit code.

Creating Interactive Scripts

- Use read command to prompt user for input.

read variable_name

example,
```
read value1
echo $value1
```

Use echo -n before read to ask the user to input from keyboard.
-n suppresses the newline character, so cursor does not move to next line.

example
```
echo -n "Enter your favorik hobby"
read hobby
echo "You entered:" $hobby
```

Options with read:

-p prompt	Use to display prompt as a text message to prompt user to input; alternative to echo -n
-s	Suppres characters; good for inputting passwords
-t timeout	Used to time out or expire the command once timeout seconds have exceeded.

Example
```
read -t 3 -s -p "Enter password:" password
echo $password
```

- timeout of 3 seconds
- Suppresses characters when user enters from keyboard
- prompts the user to enter the password.

readonly can be used to protect a variable.

Once a value is read, cannot be changed. Example uses, tax rates, zip codes, etc.

```
read -p "Enter zip code" zip
echo "Zip code is " $zip
readonly Zip
read -p " Attempting to change Zip code:" Zip
echo "Zip code is" $Zip
```

Debugging

- Trouble Shooting errors that may occur during the execution of a script.

- Use echo to show the value in a variable or parameter, but can not show very easily the flow in a script.

Use the set command

- -v option prints each line as it is read
- -x option displays the command and its arguments.

`Set -xv` ← turns ON the options

`set +xv` ← turns OFF the options

Example.

```
Set -xv
read -p "Enter Zip code" Zip
echo "Zip code is" $Zip
read only Zip
read -p "Attempting to change Zip" Zip
echo "Zip code is" $Zip
set +xv
```

As the script runs,
++ symbols appear to the LEFT of
the statement as it is executing