Command Line Programming

Edited from Stephen Mitchell - 2020

Part 1:

Directory Commands:

- cd, mkdir, rmdir, pwd, ls

File Commands:

- touch, cat, file, cp, mv, rm, head, tail, less, more

Redirections:

- *command < file*, *command <(command)*, *command > file*, *command >> file*, *command | command*

Work along?

http://bit.ly/2GKDnxj

https://bellard.org/jslinux/vm.html?cpu=riscv64&url=https://bellard.org/jslinux/buildroot-riscv64.cfg&mem=256

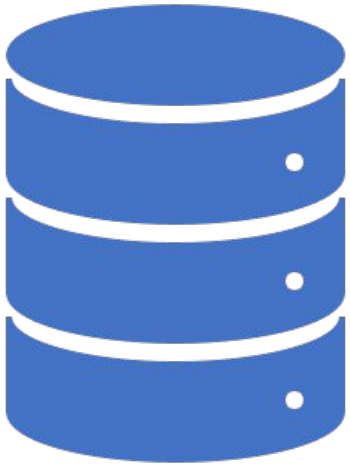
```
Loading...

Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls -l
total 24
-rw-r--r--    1 root     root           113 Sep  9 13:26 bench.py
-rw-r--r--    1 root     root           185 Sep  9 13:26 hello.c
-rw-r--r--    1 root     root           206 Sep  9 13:26 readme.txt
-rw-r--r--    1 root     root          8256 Sep  9 13:26 rv128test.bin
[root@localhost ~]#
```

Paste Here



Directory Commands

- Directory Commands:
  - ls
  - cd
  - mkdir
  - rmdir
  - pwd

The list command: `ls`

https://ss64.com/bash/ls.html

List contents of current directory:-

```
ls .
```

List contents of the parent directory:-

```
ls ..
```

And this?

```
ls ../root
```

```
[root@localhost ~]# ls .
bench.py        hello.c        readme.txt     rv128test.bin
[root@localhost ~]# ls ..
bin      etc      lib      linuxrc  mnt      proc     run      sys      usr
dev      home     lib64    media    opt      root     sbin     tmp      var
[root@localhost ~]# ls ../root
bench.py        hello.c        readme.txt     rv128test.bin
[root@localhost ~]#
```

Change Directory: cd

https://ss64.com/bash/cd.html

- Change the current working directory to a specified folder.
- If directory is not specified, change to ‘home’ directory for user.

```
cd [options] [directory]
e.g.
    cd myfolder
    ls
    cd
    Ls
```

```
[root@localhost ~]# cd myfolder
[root@localhost myfolder]# ls
[root@localhost myfolder]# cd
[root@localhost ~]# ls
bench.py        hello.c        myfolder        readme.txt      test
hello           long           q               rv128test.bin
[root@localhost ~]#
```

Make Directory: `mkdir`

https://ss64.com/bash/mkdir.html

- Create a new folder or folders unless they already exist.
- **`mkdir`** creates the standard entries . (dot) for the current folder and .. (dot dot) for its parent

```
mkdir [options] folder...
e.g.
    mkdir "name with spaces"
    mkdir name_with_no_spaces
```

```
[root@localhost ~]# mkdir "a folder"
[root@localhost ~]# ls
a folder        hello          long
bench.py        hello.c        myfolder
[root@localhost ~]#
```

Remove Directory: `rmdir`

https://ss64.com/bash/rmdir.html

- Only works if the folders are empty

```
rmdir [options] folder(s)...
e.g.
    rmdir myfolder "a folder"
```

```
[root@localhost ~]# ls
a folder        hello           long            q               rv128test.bin
bench.py        hello.c         myfolder        readme.txt      test
[root@localhost ~]# rmdir myfolder "a folder"
[root@localhost ~]# ls
bench.py        hello.c         q               rv128test.bin
hello           long            readme.txt      test
[root@localhost ~]#
```

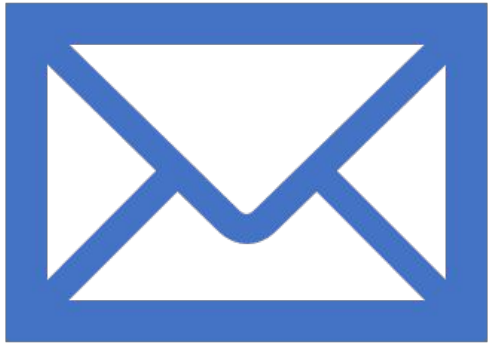
Print Working Directory: pwd

https://ss64.com/bash/pwd.html

- Options
  - `-P` : The pathname printed will not contain symbolic links.
  - `-L` : The pathname printed can contain symbolic links.
- The default action is to show the current folder as an absolute path

```
pwd [-LP]
e.g.
    pwd
```

```
mike@ME-Ubuntu:~/Desktop/Repos/sushigo$ pwd
/home/mike/Desktop/Repos/sushigo
mike@ME-Ubuntu:~/Desktop/Repos/sushigo$
```



File Commands

- File Commands:
  - touch
  - cat
  - file
  - cp
  - mv
  - rm
  - head
  - tail
  - less
  - more

Touch a file: `touch`

https://ss64.com/bash/touch.html

- Change file timestamps, change the access and/or modification times of the specified files
- Creates a file, if it does not exist or updates it to current time stamps

```
touch [options] file(s)...
e.g.
    touch newfile
```

touch example

```
[root@localhost ~]# touch newfile
[root@localhost ~]# ls
bench.py        hello.c         newfile         readme.txt      test
hello           long            q               rv128test.bin
[root@localhost ~]# ls -l
total 44
-rw-r--r--    1 root     root          113 Sep  9 13:26 bench.py
-rwxr-xr-x    1 root     root         8008 Feb 28 11:21 hello
-rw-r--r--    1 root     root          185 Sep  9 13:26 hello.c
-rw-r--r--    1 root     root          824 Feb 28 16:23 long
-rw-r--r--    1 root     root            0 Feb 28 18:04 newfile
-rw-r--r--    1 root     root          207 Feb 28 16:30 q
-rw-r--r--    1 root     root          206 Sep  9 13:26 readme.txt
-rw-r--r--    1 root     root         8256 Sep  9 13:26 rv128test.bin
-rw-r--r--    1 root     root           13 Feb 28 11:54 test
[root@localhost ~]# touch newfile
[root@localhost ~]# ls -l
total 44
-rw-r--r--    1 root     root          113 Sep  9 13:26 bench.py
-rwxr-xr-x    1 root     root         8008 Feb 28 11:21 hello
-rw-r--r--    1 root     root          185 Sep  9 13:26 hello.c
-rw-r--r--    1 root     root          824 Feb 28 16:23 long
-rw-r--r--    1 root     root            0 Feb 28 18:05 newfile
-rw-r--r--    1 root     root          207 Feb 28 16:30 q
-rw-r--r--    1 root     root          206 Sep  9 13:26 readme.txt
-rw-r--r--    1 root     root         8256 Sep  9 13:26 rv128test.bin
-rw-r--r--    1 root     root           13 Feb 28 11:54 test
[root@localhost ~]#
```

The **cat** Command

https://ss64.com/bash/cat.html

- The concatenate command.
  - Concatenates FILEs and prints them to **stdout.**
  - When file is '**-**' read standard input.

cat *[options] [filenames] [-] [filenames]*

e.g.

```
cat readme.txt hello.c - > concatenated.txt
Hello! <ENTER>
Another line of text. <ENTER>
<CTRL>+D
```

```
[root@localhost ~]# cat concatenated.txt
Some tests:

- Compile hello.c:

  gcc hello.c -o hello
  ./hello

- Compute 1000 digits of pi:

  tinypi 1000 pi.txt
  cat pi.txt

- Run the 128 bit version of riscvemu:

  riscvemu128 -m 16 rv128test.bin
/* This C source can be compiled with:
   gcc -o hello hello.c
*/
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
Hello!
Another line of text.
[root@localhost ~]#
```

cat Abuse

- Useless use of cat
  - Command line constructs that use cat where other methods would normally be used.

e.g.

```
cat readme.txt | sort -r > sorted.txt
```

Could be written:

```
sort -r readme.txt > sorted.txt
```

Determine File Type: `file`

https://ss64.com/bash/file.html

- Determines type of the provided file.

`file` ***`filename`***

(e.g. using Ubuntu18 on Windows 10)

```
steve@DESKTOP-PU5I5UD:~$ ls
codetest  test
steve@DESKTOP-PU5I5UD:~$ cat test
clear
ls -l

steve@DESKTOP-PU5I5UD:~$ file test
test: ASCII text
steve@DESKTOP-PU5I5UD:~$
```

Copy: cp

https://ss64.com/bash/cp.html

- Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY
- Copy one or more files.

```
cp [options] source destination
cp [options] source... directory
```

e.g.

```
cp -v readme.txt myfolder
```

```
[root@localhost ~]# cp -v readme.txt myfolder
'readme.txt' -> 'myfolder/readme.txt'
[root@localhost ~]# ls -l myfolder
total 4
-rw-r--r--    1 root     root           206 Feb 28 16:40 readme.txt
[root@localhost ~]#
```

Move: `mv`

https://ss64.com/bash/mv.html

- Move or rename files or directories
- If the last argument is an existing directory, **`mv`** moves the other given files into that directory
- If only two file names are given, it renames the first as the second

```
mv [options] source destination
mv [options] source... directory
```

e.g.

mv myfolder/readme.txt newreadme.txt

```
[root@localhost ~]# mv myfolder/readme.txt newreadme.txt
[root@localhost ~]# ls
bench.py        hello.c         myfolder        q
hello           long            newreadme.txt   readme.txt
[root@localhost ~]# ls myfolder
[root@localhost ~]#
```

Remove: `rm`

https://ss64.com/bash/rm.html

- Removes or deletes files and links
- Does not remove directories without using **`-r`** option
- See **`shred`** for data safe deletion

```
rm [options] filename...
e.g.
    rm newreadme.txt
```

Show head of files: head

https://ss64.com/bash/head.html

- Output the first part of files, prints the first part (10 lines by default) of each file

head [*options*] [*file(s)...*]

e.g.

head *.c *.txt

```
[root@localhost ~]# head *.c *.txt
==> hello.c <==
/* This C source can be compiled with:
   gcc -o hello hello.c
*/
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;

==> readme.txt <==
Some tests:

- Compile hello.c:

  gcc hello.c -o hello
  ./hello

- Compute 1000 digits of pi:

  tinypi 1000 pi.txt
[root@localhost ~]#
```

Show last part of files: `tail`

https://ss64.com/bash/tail.html

- Outputs the last part of files
- 10 lines (by default) of each file
- **`tail`** reads from the standard input if no files are given or if given a file of '–'

```
tail [options] [file(s)]...
e.g.
    tail *.c *.txt
```

```
[root@localhost ~]# tail *.c *.txt
==> hello.c <==
   gcc -o hello hello.c
*/
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}

==> readme.txt <==
  ./hello

- Compute 1000 digits of pi:

  tinypi 1000 pi.txt
  cat pi.txt

- Run the 128 bit version of riscvemu:

  riscvemu128 -m 16 rv128test.bin
[root@localhost ~]#
```

One screen at a time: `more`

https://ss64.com/bash/more.html

- Display output one screen at a time.
- The command **`less`** provides extensive enhancements.

```
more [-dlfpcsu] [-num] [+/ pattern] [+ linenum] [file ...]
e.g.
      cat readme.txt > long
      cat readme.txt >> long
      cat readme.txt >> long
      cat readme.txt >> long
      more long
```

```
Some tests:

- Compile hello.c:

  gcc hello.c -o hello
  ./hello

- Compute 1000 digits of pi:

  tinypi 1000 pi.txt
  cat pi.txt

- Run the 128 bit version of riscvemu:

  riscvemu128 -m 16 rv128test.bin
Some tests:

- Compile hello.c:

  gcc hello.c -o hello
  ./hello

- Compute 1000 digits of pi:

  tinypi 1000 pi.txt
  cat pi.txt

- Run the 128 bit version of riscvemu:

--More-- (47% of 824 bytes)
```

Page through files: `less`

https://ss64.com/bash/less.html

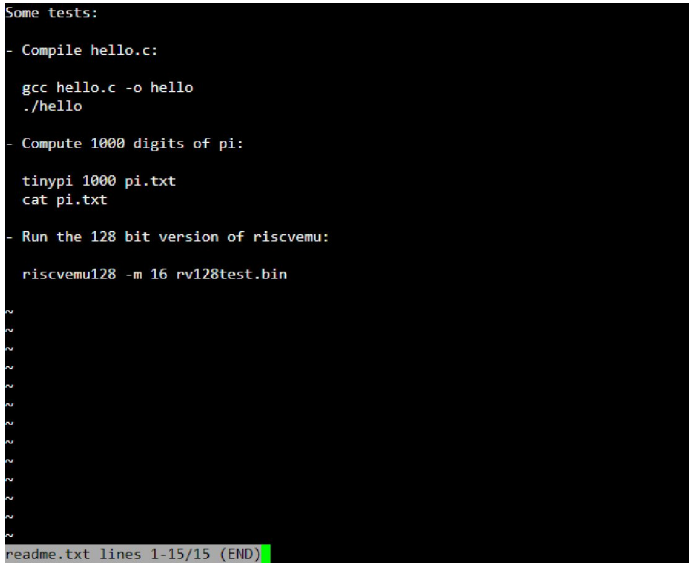
- Page through text one screenful at a time, Search through output, Edit the command line.
- Very powerful on full bash implementations.

less [*options*] *file*

command | less [*options*]

e.g.

```
less -M readme.txt
cat long | less -E
less -E long
less -?
```



Redirection

- Directing data:
  - *command < file*
  - *command <(command)*
  - *command > file*
  - *command >> file*
  - *command | command*

I/O Redirection

http://bit.ly/2VqsGDj

- There are always three default files open:
  - **stdin** (the keyboard, channel 0)
  - **stdout** (the screen, channel 1)
  - **stderr** (error messages output to the screen, channel 2).

- These, and any other open files, can be redirected.

- Redirection simply means capturing output from a file, command, program, script, or even code block within a script and sending it as input to another file, command, program, or script.

https://www.tldp.org/LDP/abs/html/io-redirection.html

[The Linux Documentation Project tldp.org]

COMMAND_OUTPUT >

- Redirect **stdout** to a file.
  - Creates the file if not present, otherwise overwrites it.

```
ls -lR > dir-tree.list
```

COMMAND_OUTPUT >>

- Redirect **stdout** to a file.
  - Creates the file if not present, otherwise appends to it.

```
ls -lR >> dir-tree.list
```

Redirect **stdout**

- 1 > filename
  - Redirect **stdout** to file "filename.“
- 1 >> filename
  - Redirect and append **stdout** to file "filename."

Redirect **stderr**

- `2 > filename`
  - Redirect stderr to file "filename.“
- `2 >> filename`
  - Redirect and append stderr to file "filename."

Redirecting **stdout**, one line at a time

```
LOGFILE=script.log
echo "This statement is sent to the log file, \"$LOGFILE\"." 1>$LOGFILE
echo "This statement is appended to \"$LOGFILE\"." 1>>$LOGFILE
echo "This statement is also appended to \"$LOGFILE\"." 1>>$LOGFILE
echo "This statement is echoed to stdout, and will not appear in \"$LOGFILE\"."

clear
cat script.log
```

```
[root@localhost ~]# cat script.log
This statement is sent to the log file, "script.log".
This statement is appended to "script.log".
This statement is also appended to "script.log".
[root@localhost ~]#
```

Redirecting **stderr**, one line at a time.

```
ERRORFILE=script.errors

bad_command1 2>$ERRORFILE       #  Error message sent to $ERRORFILE.
bad_command2 2>>$ERRORFILE      #  Error message appended to $ERRORFILE.
bad_command3                    #  Error message echoed to stderr.

clear
cat script.errors
```

```
[root@localhost ~]# cat script.errors
sh: bad_command1: not found
sh: bad_command2: not found
[root@localhost ~]#
```

Redirect **stdout** & **stderr**

`&>FILENAME`

- Redirects both **stdout** and **stderr** to file "filename.“
  - This operator is now functional, as of Bash 4, final release.

```
echo "hey – this is an output, where does it go?" &>output.txt
echo "oops!" >>output.txt | Bad_command1 2>>output.txt
```

```
[root@localhost ~]# echo "hey – this is an output, where does it go?" &>output
.txt
[root@localhost ~]# echo "oops!" >>output.txt | Bad_command1 2>>output.txt
[root@localhost ~]# cat output.txt
hey – this is an output, where does it go?
oops!
sh: Bad_command1: not found
[root@localhost ~]#
```

< FILENAME

```
0< FILENAME
< FILENAME
```

- Accept input from a file.
  - Companion command to ">", and often used in combination with it.

```
grep *.c < dir-tree.list
```

```
[root@localhost ~]# grep *.c < dir-tree.list
-rw-r--r--    1 root     root          185 Sep  9 13:26 hello.c
-rw-r--r--    1 root     root          185 Sep  9 13:26 hello.c
[root@localhost ~]#
```

The Pipe “|”

- Pipe “|” (represented by the vertical bar) is used send output to another program.

e.g.

```
cat | sort > sortdata.txt
line 1
line 2
line 0
```

<CTRL> + D

```
clear
cat sortdata.txt
```

```
[root@localhost ~]# cat sortdata.txt
line 0
line 1
line 2
[root@localhost ~]#
```

Part 2

Common Flags:

- -a, -R, -r, -t, -S, -l, -1, -m, -Q

Permissions:

- chmod *nnn file*, chmod –R *nnn folder*, chown *user*:*group file*
- read (r) [4], write (w) [2], execute (x) [1]

Wild Cards:

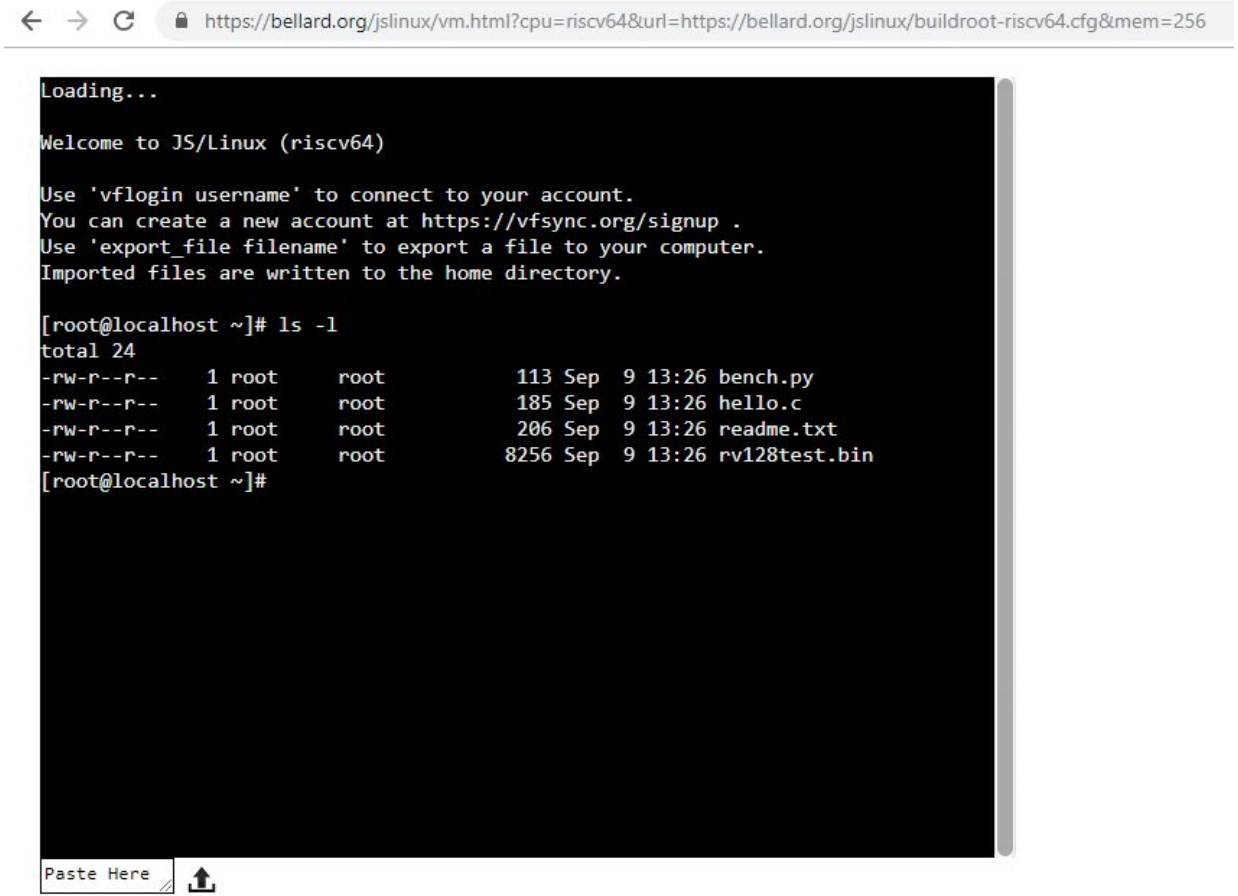
- ?, *, [], {}

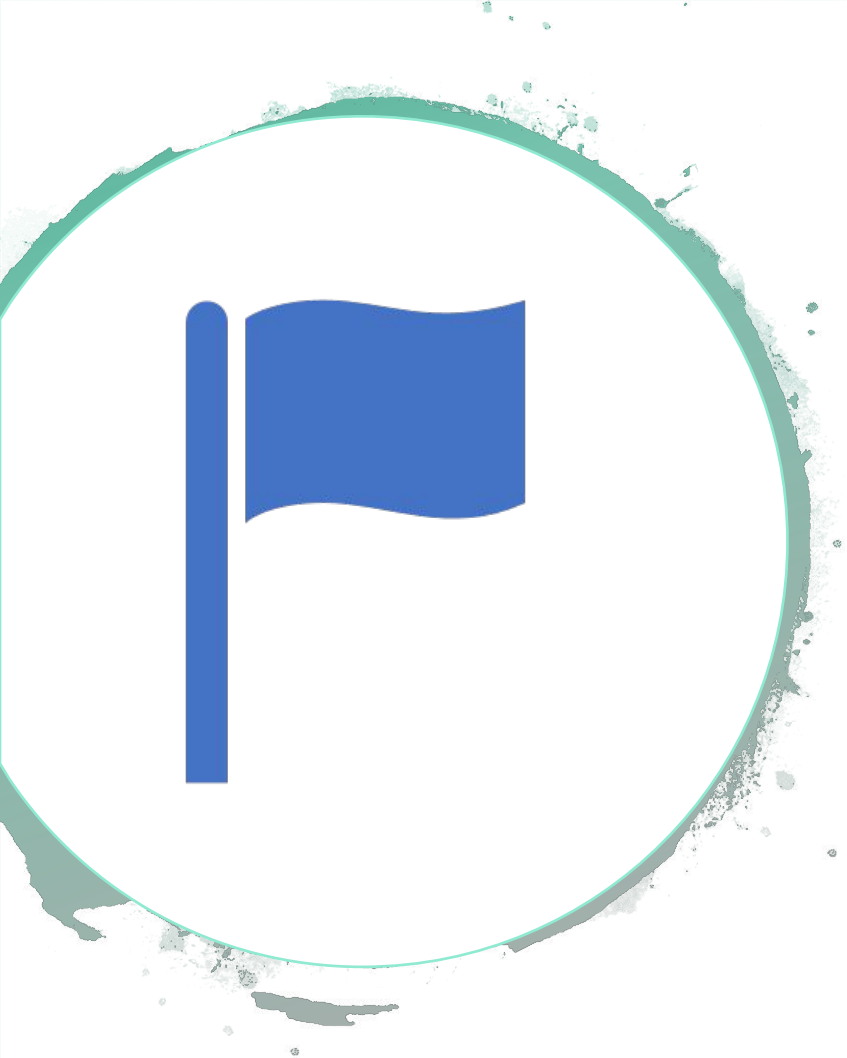
Process Control:

- ***<CTRL>+c*, *<CTRL>+z*, *<CTRL>+a*, *<CTRL>+e*, *<CTRL>+u*, *<CTRL>+k*, *<CTRL>+d***

Work along?

http://bit.ly/2GKDnxj





Common Flags

- Common Flags:
  - -a
  - -R
  - -r
  - -t
  - -S
  - -l
  - -1
  - -m
  - -Q

Case is Sensitive

- Be AWARE, in Linux the case of the character is normally significant.
- Not all lower case flags are the same as their upper case equivalent.
- But, confusingly, some are…
- Where there are obvious ‘confusions’ I have included the similar flag to explain the difference.
- Often the file management commands, such as ‘**`ls`**’, have the most typical flags/options. I have focused on the ‘**`ls`**’ description as different commands can vary.

-a, --all / -A, --almost-all

- -a, --all
  - Show **all** information or operate on all arguments.
  - May include hidden files.
- -A, --almost-all
  - Except for '.' and '..' (implied).

e.g.

```
ls -a
ls -A
```

-r, --reversive / -R, --recursive

- `-r, --reversive`
  - Reverse order whilst sorting.
- -R / --Recursive
  - Include subdirectories, i.e. down the directory tree.

e.g.

```
ls -r
rm –R myDirectory
```

-t / -T, --tabsize=*COLUMNS*

- -t
  - Sort by modification time.
- -T, --tabsize=*COLUMNS*
  - Assume tab stops at each *COLUMNS* instead of 8
  - i.e. replace *COLUMNS* with a value other than 8

e.g.

```
ls -t
ls -T4
```

-s, --size

- Print the size of each file, in blocks.

e.g.

```
ls -S
```

-S, --sort=*WORD*

- -S
  - Sort by file size.
- --sort=*WORD*
  - Where *WORD* =
    - time -t, version -v, status -c, size -S, extension -X,
    - none -U, atime -u, access -u, use -u

e.g.

```
ls -S
ls --sort=extension
```

`-l`

- `-l`
  - Use a long listing format.

e.g.

```
ls -l
```

-1

- List one file per line.

e.g.

```
ls -1
```

-m

- Fill width with a comma separated list of entries.

e.g.

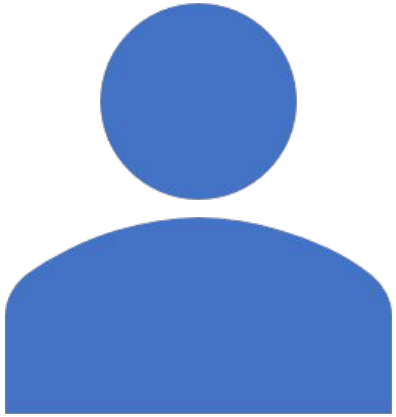
```
ls -m
```

`-q, --quiet, -Q`

- `-q`
  - Operate commands quietly.
  - --hide-control-chars
    - Print ? Instead of non graphic characters
- `--quiet`
  - Suppress stdout.
- `-Q`
  - --quote-name
    - Enclose entry names in double quotes

e.g.

```
ls -Q
```



Permissions

- Permissions:
  - chmod *nnn file*
  - chmod –R *nnn folder*
  - chown *user*:*group file*
    - read (r) [4]
    - write (w) [2]
    - execute (x) [1]

Ownership

- Linux supports a large number of users, consequently it needs to keep track of who and how users can access files and folders.
- These rules are called permissions.
- There are three common types of users with permissions.
  - User
    - These apply to a single user, it gives them special access. The user is the ‘**owner**’ of the file.
  - Group
    - Apply to a single group of users; the ‘**owning group**’.
  - Other
    - All other users

https://www.computerhope.com/unix/uchown.htm

chmod

- https://ss64.com/bash/chmod.html
- Change access permissions, **ch**ange **mod**e.
- Supports ‘numeric mode’ and ‘symbolic mode’.

```
chmod nnn file
```

e.g.

```
chmod 775 readmet.txt
```

read, write, execute

- chmod 400 filename - Read by owner
- chmod 040 filename - Read by group
- chmod 004 filename - Read by other
- chmod 200 filename - Write by owner
- chmod 020 filename - Write by group
- chmod 002 filename - Write by other
- chmod 100 filename - Execute by owner
- chmod 010 filename - Execute by group
- chmod 001 filename - Execute by other

chmod 444 filename – adds read to all 3.

chmod -R

- change files and directories recursively.

`chmod -R nnn folder`

e.g.

```
mkdir testFolder
touch testFolder/inside.txt
ls ./testFolder -l
chmod -R 600 testFolder
ls ./testFolder -l
```

```
[root@localhost ~]# mkdir testFolder
[root@localhost ~]# touch testFolder/inside.txt
[root@localhost ~]# ls ./testFolder -l
total 0
-rw-r--r--    1 root     root            0 Mar  4 11:59 inside.txt
[root@localhost ~]# chmod -R 600 testFolder
[root@localhost ~]# ls ./testFolder -l
total 0
-rw-------    1 root     root            0 Mar  4 11:59 inside.txt
[root@localhost ~]#
```

chown

- https://ss64.com/bash/chown.html
- Change owner, change the user and/or group ownership of each given File to a new Owner.
- `chown` can also change the ownership of a file to match the user/group of an existing reference file.
- The ‘:’ is used to differentiate between the individual **user** and **group**.

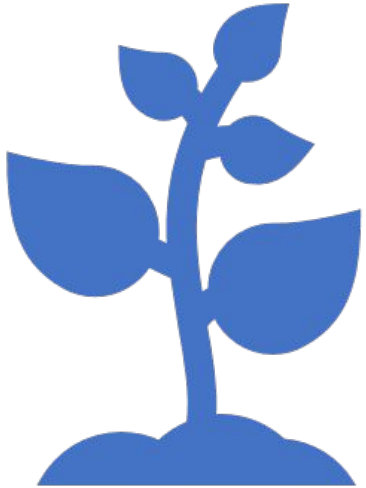
chown

chown *filename* [*user*[:*group*]]

e.g.

```
groupadd new_group
ls -l
chown hello.c root:new_group
ls -l
```

```
root@tryit-holy:~# groupadd new_group
root@tryit-holy:~# ls -l
total 0
-rw-r--r-- 1 root root 0 Mar  4 14:19 hello.c
root@tryit-holy:~# chown root:new_group hello.c
root@tryit-holy:~# ls -l
total 0
-rw-r--r-- 1 root new_group 0 Mar  4 14:19 hello.c
root@tryit-holy:~#
```



Wild Cards

- Wild Cards:
  - *
  - ?
  - []
  - {}

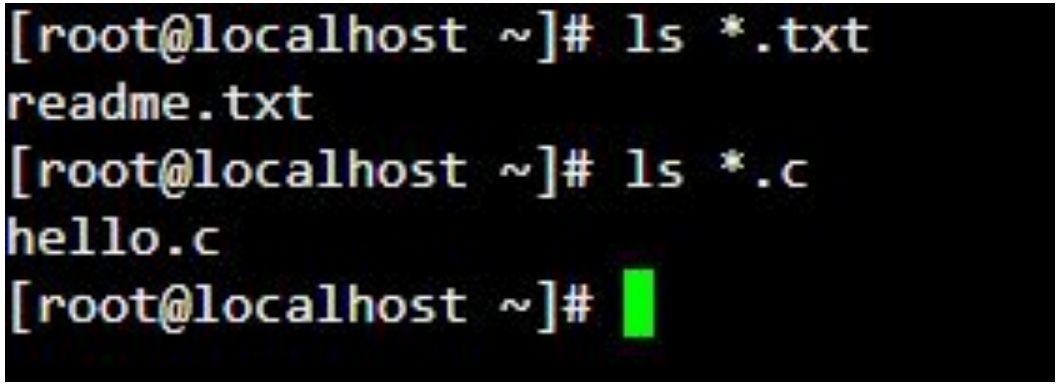
*

- Match all characters.

e.g.

```
ls *.txt

ls *.c
```



?

- Match for a single character.

e.g.

```
cat *.?

cat *.??
```

```
[root@localhost ~]# cat *.?
/* This C source can be compiled with:
   gcc -o hello hello.c
*/
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
[root@localhost ~]# cat *.??
def fib(n):
    if n < 2:
        return n
    return fib(n-2) + fib(n-1)

n = 25
print "fib(", n, ")= ", fib(n)
[root@localhost ~]#
```

[ ]

- Match range of values.

e.g.

```
touch test1.txt
touch test2.txt
touch test3.txt
ls test[1-2].*
rm test1.txt
ls
```

```
[root@localhost ~]# touch test1.txt
[root@localhost ~]# touch test2.txt
[root@localhost ~]# touch test3.txt
[root@localhost ~]# ls
bench.py        long            rv128test.bin  test2.txt      testFolder
hello.c         readme.txt      test1.txt      test3.txt
[root@localhost ~]# ls test[1-2].*
test1.txt  test2.txt
[root@localhost ~]# rm test1.txt
[root@localhost ~]# ls
bench.py        long            rv128test.bin  test3.txt
hello.c         readme.txt      test2.txt      testFolder
[root@localhost ~]#
```

{ }

- Match list of values.

e.g.

```
touch test1.txt
touch test2.txt
touch test3.txt
ls
ls text{1,3}.*
```

```
steve@DESKTOP-PU5I5UD:~/codetest/example$ touch text1.txt
steve@DESKTOP-PU5I5UD:~/codetest/example$ touch text2.txt
steve@DESKTOP-PU5I5UD:~/codetest/example$ touch text3.txt
steve@DESKTOP-PU5I5UD:~/codetest/example$ ls
text1.txt  text2.txt  text3.txt
steve@DESKTOP-PU5I5UD:~/codetest/example$ ls text{1,3}.*
text1.txt  text3.txt
steve@DESKTOP-PU5I5UD:~/codetest/example$
```

Process Control

- Process Control:
  - ***<CTRL>+c***
  - ***<CTRL>+z***
  - ***<CTRL>+a***
  - ***<CTRL>+e***
  - ***<CTRL>+k***
  - ***<CTRL>+u***
  - ***<CTRL>+d***

<CTRL>+C

- Kill whatever you are running

e.g.

```
cat > empty
<CTRL>+C
```

```
[root@localhost ~]# cat > empty
^C
[root@localhost ~]#
```

<CTRL>+Z

- Puts whatever you are running into a suspended background process.
- fg command restores it.

`fg` *`process_number`*

e.g.

```
cat > empty
<CTRL>+Z
fg [1]
```

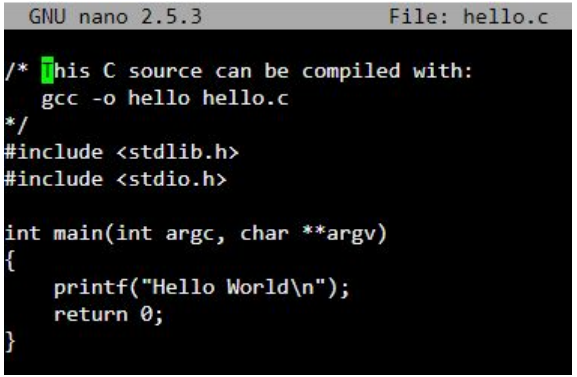
```
root@tryit-holy:~# cat > empty
^Z
[1]+  Stopped                 cat > empty
root@tryit-holy:~# fg 1
cat > empty
root@tryit-holy:~#
```

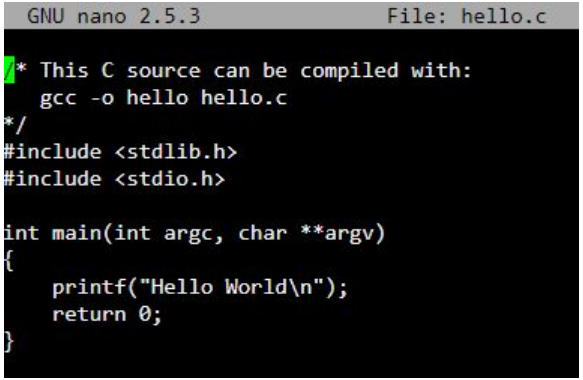
<CTRL>+A

- Go to the beginning of the line you are currently typing on

e.g.

```
nano hello.c
<RIGHT><RIGHT><RIGHT>
<CTRL>+A
```





<CTRL>+E

- Go to the end of the line you are currently typing on.

<CTRL>+E

```
  GNU nano 2.5.3                    File: hello.c

/* This C source can be compiled with:
   gcc -o hello hello.c
*/
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
```

<CTRL>+K

- Clear the line after the cursor and paste into the buffer.

<CTRL>+K

```
  GNU nano 2.5.3                    File: hello.c

  gcc -o hello hello.c
*/
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
```

<CTRL>+U

- Pastes the buffer.

<CTRL>+U

```
  GNU nano 2.5.3                File: hello.c

/* This C source can be compiled with:
   gcc -o hello hello.c
*/
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
```

<CTRL>+D

- Delete
- Exit the current shell

```
[root@localhost ~]# cat > test.txt
enter first line
enter second line
[root@localhost ~]#
```

e.g.

```
cat > test.txt <ENTER>
enter first line <ENTER>
enter second line <ENTER>
<CTRL>+D

<CTRL>+D (in nano, deletes)
```

```
  GNU nano 2.5.3                    File: hello.c

/* This C source can be compiled with:
hello hello.c
*/
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
```

Bonus Material

- write
- exec
- man
- ?, -h, --help
- <TAB>
- <UP> <DOWN>

write

- Send a message to another user.
- Write allows you to communicate with other users, by copying lines from your terminal to theirs.
- When you run the write command, the user you are writing to gets a message of the form:

```
Message from yourname@yourhost on yourtty at hh:mm ...
```

- Any further lines you enter will be copied to the specified user's terminal. If the other user wants to reply, they must run write as well.
- When you are done, type an end-of-file or interrupt character. The other user will see the message EOF indicating that the conversation is over.

`write` *`username`*

exec

- https://ss64.com/bash/exec.html
- Execute a command
- If command is supplied, it replaces the shell without creating a new process. If no command is specified, redirections can be used to affect the current shell environment.

```
exec [-cl] [-a name] [command [arguments]]
```

`-c` Causes command to be executed with an empty environment.
`-l` Place a dash at the beginning of the zeroth argument passed to command. (This is what the login program does.)
`-a` The shell passes name as the zeroth argument to command.

exec

Running Ubuntu Bash on your windows PC, then running…

```
exec nano <ENTER>
```

…means ***nano*** has taken over the thread that the bash shell was running in.

<CTRL>+X

…doesn't just exit ***nano***, it closes the entire window!

man

- https://ss64.com/bash/man.html
- Man displays help pages.
- For more information '***man***' ***man***

```
man man
```

```
MAN(1)                                   Manual pager utils                                   MAN(1)

NAME
       man - an interface to the on-line reference manuals

SYNOPSIS
       man  [-C  file]  [-d]  [-D]  [--warnings[=warnings]] [-R encoding] [-L locale] [-m system[,...]] [-M path] [-S
       list] [-e extension] [-i|-I] [--regex|--wildcard] [--names-only] [-a]  [-u]  [--no-subpages]  [-P  pager]  [-r
       prompt] [-7] [-E encoding] [--no-hyphenation] [--no-justification] [-p string] [-t] [-T[device]] [-H[browser]]
       [-X[dpi]] [-Z] [[section] page[.section] ...] ...
       man -k [apropos options] regexp ...
       man -K [-w|-W] [-S list] [-i|-I] [--regex] [section] term ...
       man -f [whatis options] page ...
       man -l [-C file] [-d] [-D] [--warnings[=warnings]] [-R encoding] [-L locale] [-P pager] [-r prompt]  [-7]  [-E
       encoding] [-p string] [-t] [-T[device]] [-H[browser]] [-X[dpi]] [-Z] file ...
       man -w|-W [-C file] [-d] [-D] page ...
       man -c [-C file] [-d] [-D] page ...
       man [-?V]

DESCRIPTION
       man  is the system's manual pager.  Each page argument given to man is normally the name of a program, utility
       or function.  The manual page associated with each of these arguments is then found and displayed.  A section,
       if  provided,  will direct man to look only in that section of the manual.  The default action is to search in
       all of the available sections following a pre-defined order ("1 n l 8 3 2 3posix 3pm 3perl 3am 5 4 9 6  7"  by
       default,  unless  overridden by the SECTION directive in /etc/manpath.config), and to show only the first page
       found, even if page exists in several sections.

       The table below shows the section numbers of the manual followed by the types of pages they contain.

       1   Executable programs or shell commands
       2   System calls (functions provided by the kernel)
       3   Library calls (functions within program libraries)
       4   Special files (usually found in /dev)
       5   File formats and conventions eg /etc/passwd
       6   Games
       7   Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
       8   System administration commands (usually only for root)
       9   Kernel routines [Non standard]
 Manual page man(1) line 1 (press h for help or q to quit)
```

-?, --help or -h

- Some commands offer help outside of using man.
- -? May give a clue as what to do.
- -h or -help should give good essential information on how to use a command or application.

e.g.

```
ls --help
```

```
List directory contents

        -1      One column output
        -a      Include entries which start with .
        -A      Like -a, but exclude . and ..
        -C      List by columns
        -x      List by lines
        -d      List directory entries instead of contents
        -L      Follow symlinks
        -H      Follow symlinks on command line
        -R      Recurse
        -p      Append / to dir entries
        -F      Append indicator (one of */=@|) to entries
        -l      Long listing format
        -i      List inode numbers
        -n      List numeric UIDs and GIDs instead of names
        -s      List allocated blocks
        -e      List full date and time
        -h      List sizes in human readable format (1K 243M 2G)
        -r      Sort in reverse order
        -S      Sort by size
        -X      Sort by extension
        -v      Sort by version
        -c      With -l: sort by ctime
        -t      With -l: sort by mtime
        -u      With -l: sort by atime
        -w N    Assume the terminal is N columns wide
        --color[={always,never,auto}]   Control coloring
[root@localhost ~]#
```

TAB

- Pressing tab completes file names or paths at the command line.

e.g.

cat rea<*TAB*>

TAB

- Pressing tab completes file names or paths at the command line.

e.g.

```
cat readme.txt
```

UP and DOWN keys

- Pressing the UP or DOWN arrow key moves through the history of previously entered commands at your command prompt.

e.g.

```
gcc hello.c -o hello <ENTER>
./hello <ENTER>
Hello World
<UP>
```

UP and DOWN keys

- Pressing the UP or DOWN arrow key moves through the history of previously entered commands at your command prompt.

e.g.

```
gcc hello.c -o hello <ENTER>
./hello <ENTER>
Hello World
./hello <UP>
```

UP and DOWN keys

- Pressing the UP or DOWN arrow key moves through the history of previously entered commands at your command prompt.

e.g.

```
gcc hello.c -o hello <ENTER>
./hello <ENTER>
Hello World
gcc hello.c –o hello
```