



LINUX SHELL & COMMANDS

IAC 16/17

WHY LINUX SHELL & COMMANDS?

Before graphical interfaces were a common place, computational systems relied on text-based terminals to edit (text) files, and control program execution. Even though they may seem to be a thing from the past, they are very powerful because they allow us to create complex and highly customized applications at the cost of simpler and smaller ones. You can think of it like software building-blocks. Many advanced and high-level programming, or scripting languages, follow such approach.

Throughout your degree at IST, and very likely while working as professional in the future, you'll be faced with the need to produce automated tasks, "fine-tuned" applications and other programming experiments that don't have (and don't need) a graphical interface. For such applications shell programming is a very powerful and flexible tool that you should consider learning and using.

INTRODUCTION

The best way to learn the Linux command line is as a series of small, easy to follow steps. This tutorial is organized as such, with each section building upon the knowledge and skills learned in the previous sections. Work through them, read them fully and practice on the command line as you go.

Each section is structured in the following format:

- An introduction outlining what you will learn in that section.
- Detailed material including examples.
- A set of activities to help you solidify your knowledge and skills.

The symbol ✂ represents an exercise you should do. Treat the activities as a starting point for exploration. The further you take it, the better you will do.

This tutorial is in English on purpose, so that you get acquainted with the English terms related to the Linux shell.

SOME GENERAL SYNTAX:

- I'll refer to Linux in the following pages, whenever I do, assume I'm actually saying UNIX/Linux. Linux is an offshoot of UNIX and behaves pretty much exactly the same.
- Whenever you see **<something>**, what this means is that you are to replace this with something useful. Replace the whole thing (including the < and >). If you see something such as **<n>** then it usually means replace this with a number.
- Whenever you see **[something]** this usually means that this something is optional. When you run the command you may put in the something or leave it out.

PROBLEM SOLVING AND CREATIVE THINKING

If you wish to succeed with the Linux command line then there are two things you need, Problem solving and Creative thinking. Here are some basic pointers to help you along the way.

- **Explore and experiment.** Remember, you're learning about a set of building blocks and with them you can build almost anything. The examples you will find are intended to be an illustration of how they work, not the only thing you can do with them. I encourage you to tweak the examples and see how they behave. This will give you a much better understanding on how they work. You will have a lot of questions along the way along the lines

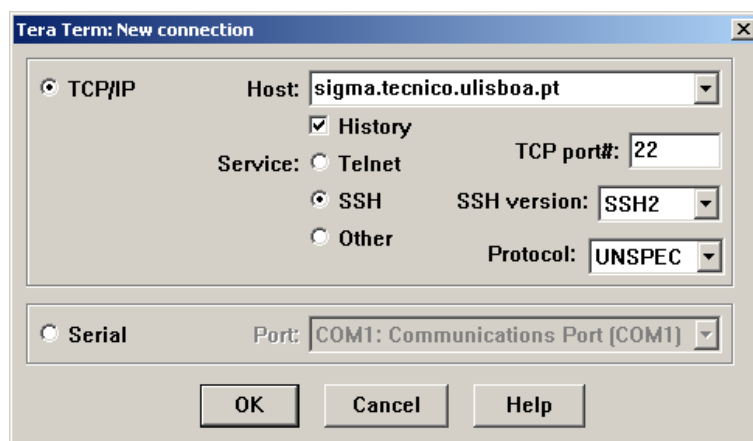
of "What if....?" and "Can I ...?" to which I say, "Give it a go and see what happens." The worst you can really get is an error message. In which case you read the error message to understand why it didn't work, then have another go. Don't hold back!

- **Read carefully** and don't skip over the fine details. I can't stress this enough. The fine details are important and are often the difference between your command working and not working. If something isn't working then re-read the material carefully and re look over what you have typed in to make sure you haven't made a silly little typo.

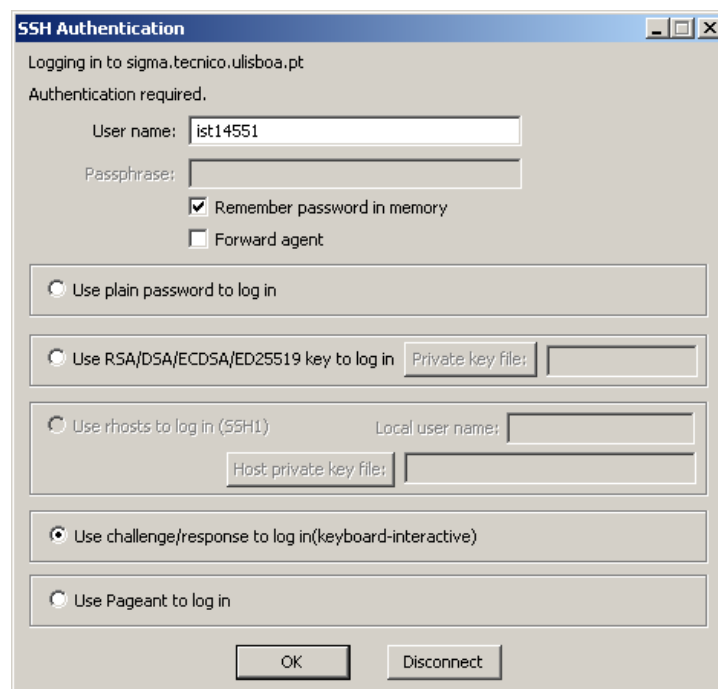
So the general approach is:

- Create a hypothesis.
- Run your command to test this hypothesis.
- Observe the output. If it is what you were expecting, great. If not then continue.
- Analyze the output and adjust your understanding accordingly.
- Rinse and repeat till you get what you are after.

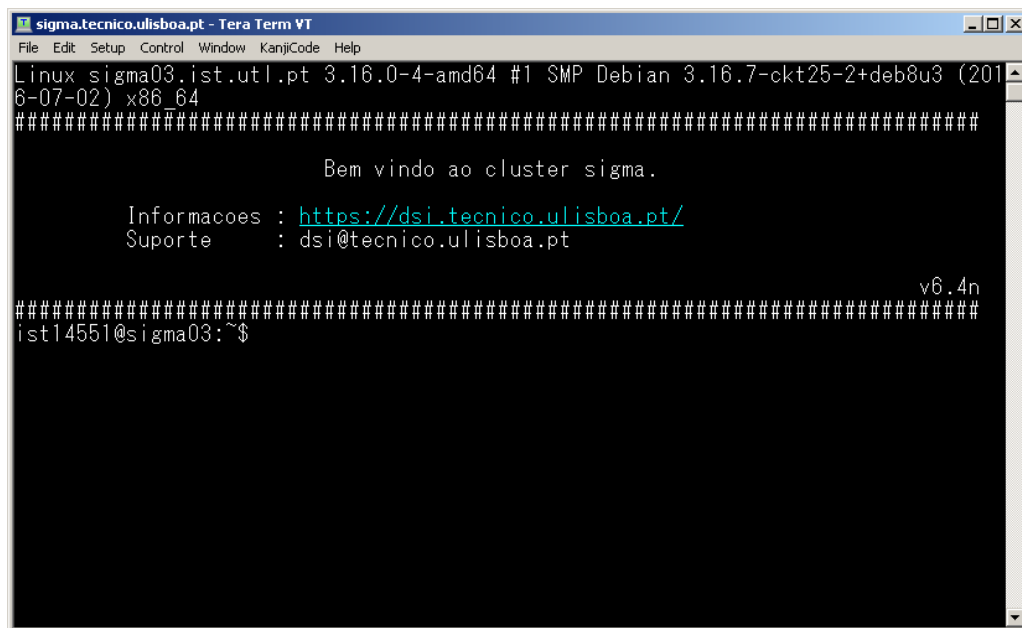
NOTE: Outside the lab, you can follow this tutorial using the sigma server available at IST. Use your favorite terminal emulator tool, e.g. TeraTerm and establish a connection as shown:



Insert your user name:



and then your password, when asked. After a successful login, you'll find the welcome screen:



```
sigma.tecnico.ulisboa.pt - Tera Term VT
File Edit Setup Control Window KanjiCode Help
Linux sigma03.ist.utl.pt 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt25-2+deb8u3 (201
6-07-02) x86_64
#####
                Bem vindo ao cluster sigma.

Informacoes : https://dsi.tecnico.ulisboa.pt/
Suporte      : dsi@tecnico.ulisboa.pt

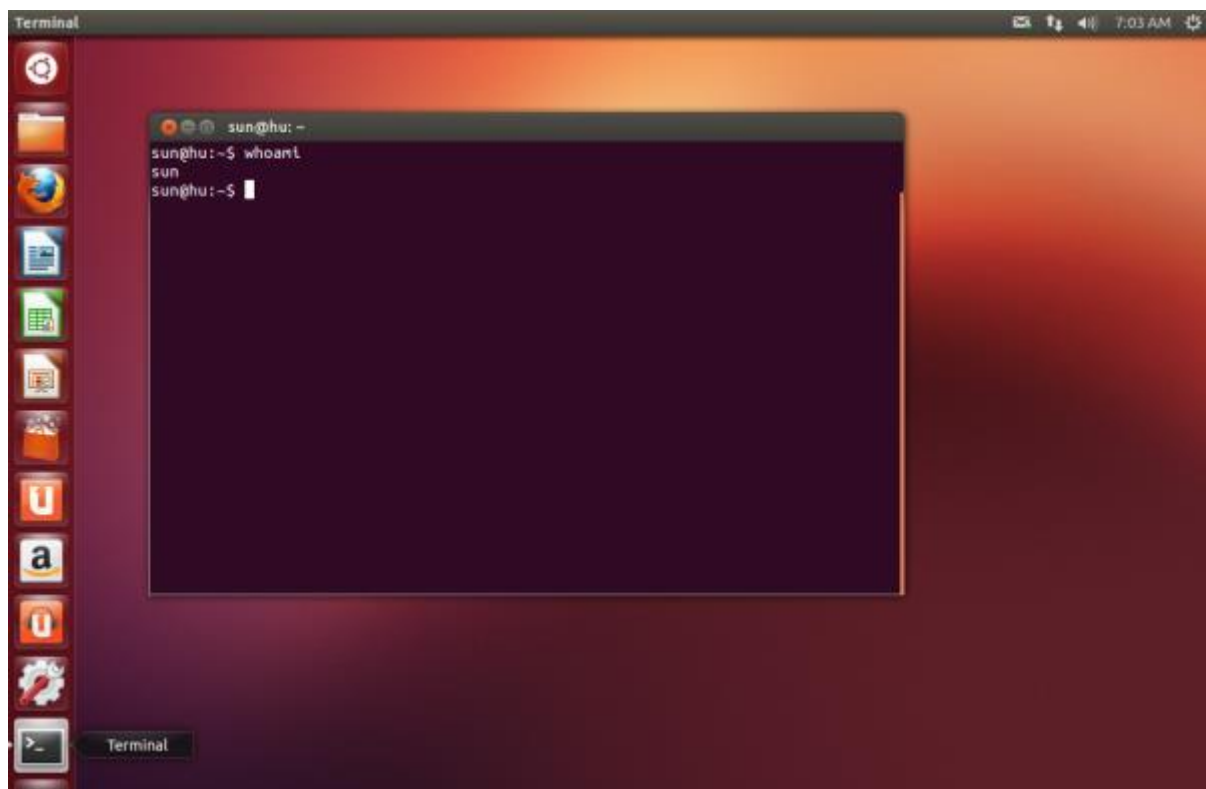
#####
v6.4n
ist14551@sigma03:~$
```

SHELLS

You can access a computer's Shell after you login. Depending on your computer's configuration you have different ways to access it.

If you're using a text-based login, either local or remote, once you login you'll be in the Shell.

If you're using a graphical interface, you need to launch it manually, usually by running the terminal or command-line under accessories or system tools. Different Linux distributions may use different names. To log out of your terminal type exit.



Once the shell starts the first thing you will see is the `user_name@computer_name$`, and this is known as prompt. In front of this line you'll enter your commands interactively.

The behavior of the terminal interface will differ slightly depending on the *shell* program that is being used. Examples:

- Bourne shell (sh):
 - Bash (bash)
 - Korn Shell (ksh)
 - Z Shell (zsh)
 - [Debian] Almquist Shell (dash/ash)
- C Shell (csh)

Depending on the shell used, some extra behaviors can be quite nifty. You can find out what shell you are using by issuing the command: `echo $SHELL`.



Exercise: start a terminal/bash shell and issue the following commands:

```
echo $SHELL
```

The commando “`echo`” will print the text passed as argument on the terminal. In this example, `$SHELL` is used to hold information about the shell in use.

Executing the command on your shell, you should get the following:

```
/bin/bash
```

You can create a file with a list of shell commands and execute it like a program to perform a task. This is called a shell script. This is in fact the primary purpose of most shells, not the interactive command line behavior.

ENVIRONMENT VARIABLES

You can have your shell to remember things for later, using environment variables via command `export`.

It is done via command `export`: `export <environment variable>=<variable value>`



Exercise: under the bash shell type the following:

```
export IAC=IAC1617
```

The terminal won't produce any output or confirmation that the command was successfully completed. You can check its value using the `echo` command as shown above:

```
ist14551@sigma03:~$ echo $IAC  
IAC1617
```

By prefixing `$` to the variable name, you can evaluate it in any command. Alternatively you can use the command

```
printenv IAC
```


Note that there's no `$` sign in this case.

Execute `printenv` without arguments, the command will display all the existent environment variables.

```
ist14551@sigma03:~$ printenv  
TERM=xterm  
SHELL=/bin/bash  
SSH_CLIENT=146.193.44.91 51213 22  
SSH_TTY=/dev/pts/0  
USER=ist14551  
IAC=IAC1617  
MAIL=/var/mail/ist14551  
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games  
PWD=/afs/ist.utl.pt/users/5/1/ist14551  
LANG=en_US.UTF-8
```

```
KRB5CCNAME=FILE:/tmp/krb5cc_14551_1kU9jJ
SHLVL=1
HOME=/afs/ist.utl.pt/users/5/1/ist14551
MATHEMATICA_HOME=/usr/local/Wolfram/Mathematica/10.2
LOGNAME=ist14551
_=/usr/bin/printenv
```

One important environment variable is PATH. It allows to execute programs under different directories other than the current directory. All paths need to be on the \$PATH variable, separated by a colon : No spaces are allowed here unless they are proceeded by the escape character \.

 Exercise: List the first 9 paths available on your PATH variable:

_____	_____	_____
_____	_____	_____
_____	_____	_____

At this point we're not interest in changing the PATH, but here are examples on how to do it:

How to create a new PATH:


```
export PATH=/home/user/bin:/usr/games
```

How to append a new path to the existing PATH variable (to avoid deleting/replacing the existing ones):

```
export PATH=$PATH:/usr/games
export PATH=$PATH:/usr/more\ games
```

INTERACTIVE HISTORY

A feature of bash and tcsh (and sometimes others) you can use the up-arrow keys to access your previous commands, edit them, and re-execute them.

 Exercise: Press up-arrow key to retrieve your last entered command, it should show `printenv` in your prompt. Now complete the command adding PATH as the argument of `printenv`:

```
ist14551@sigma01:~$ printenv PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```


You can find your command history in file located in `~/.bash_history`.

MOUSE

You can use the mouse on your shell. Obviously not to point like in a graphical environment, but to automate your input.

- LEFT-BUTTON: when dragged you can select text on your terminal.
- MIDDLE-BUTTON: copy & paste the selected text.
- RIGHT-BUTTON: paste from the clipboard or other terminal options.

As usual, different environments may have different behaviors assigned to left and right buttons.

 Exercise: Try your mouse buttons on your terminal. Don't worry about any error messages.

FILENAME COMPLETION

A feature of bash and others is that you can use the TAB key to complete a partially typed filename. For example if you have a file named `introduction-to-computer-architectures.txt` in your directory and want to edit it you can type `'vi intro'`, hit the TAB key, and the shell will fill in the rest of the name for you (provided the completion is unique).

BASH IS THE WAY COOL SHELL.

Bash will even complete the name of commands and environment variables. And if there are multiple completions, if you hit TAB twice bash will show you all the completions. Bash is the default user shell for most Linux systems.

 Exercise:

Type: `ech + <TAB> + <SPACE> + $PA + <TAB> + <ENTER>`

LOOKING FOR HELP: THE MAN COMMAND

Most of the commands have a manual page which give sometimes useful, often more or less detailed, sometimes cryptic and unfathomable descriptions of their usage.

Example:

```
man printenv
```

Shows the manual page for the `printenv` command

Press `q` to exit the man page, `SPACE` to advance one terminal screen, and up/down arrows to move the text up and down, respectively. You can further investigate the functionality of man pages by running:

```
man man
```

A simpler explanation for each command is usually available by issuing the command with the arguments `--help`.
Example:

```
printenv --help
```

 Exercise:

1. Determine who wrote the `echo` command you're using.
2. What are the options of the `echo` command? What do they do?

LISTING DIRECTORY CONTENTS:

The first set of commands that you're going to learn about are used to list the contents of a directory, navigate through the different folders of the filesystem, i.e. list files and folders stored in the hard drive or other storage devices available.

```
ls                list the contents of the current directory
ls -l             list a directory in long (detailed) format
ls -a            list the current directory including hidden files. Hidden files start with "."
ls -d            list directory entries instead of contents
ls -i            list inode of each file
ls -l            list files in long format
ls .*            list files begin with .
```


Example:

```
ls -l
drwxr-xr-x    4 cliff    user      1024 Jun 18 09:40 WAITRON_EARNINGS
-rw-r--r--    1 cliff    user      767392 Jun  6 14:28 scanlib.tar.gz
^  ^  ^  ^      ^      ^      ^      ^
|  |  |  |      |      |      |      |
```

					owner	group	size	date	time	name
					number of links to file or directory contents					
					permissions for world					
					permissions for members of group					
					permissions for owner of file: r = read, w = write, x = execute --no permission					
type of file: - = normal file, d=directory, l = symbolic link, and others...										

Open a shell and try ls using options in your current directory: -ld, -u, -g, -r, -t. Use man ls to investigate what these options do.

If you don't have/see any files on your current directory, before running the ls command, type: cd /etc

 Exercise: List all the contents of your directory sorted by modification time, from the newest to oldest. What are the arguments of the ls command to produce such output?

-rw-----	1	ist14551	14551	1560	Sep 23 08:27	.bash_history
drwxr-xr-x	2	root	root	6144	Sep 21 08:14	..
drwx--x--x	15	ist14551	14551	2048	Sep 18 23:32	.
-rw-----	1	ist14551	14551	809	Sep 18 23:32	.viminfo
-rw-r--r--	1	ist14551	14551	36	Sep 18 22:56	.bashrc
drwxr-xr-x	3	ist14551	14551	2048	Sep 18 22:54	SO1617
drwxr-xr-x	9	ist14551	14551	2048	Jul 21 20:12	old
drwxr-xr-x	2	ist14551	14551	2048	Jul 19 13:58	Desktop
drwxr-xr-x	2	ist14551	14551	2048	Jul 19 13:58	Documents
drwxr-xr-x	2	ist14551	14551	2048	Jul 19 13:58	Downloads
drwxr-xr-x	2	ist14551	14551	2048	Jul 19 13:58	Music

DIRECTORIES:

The file system, which holds all the files and folder of your computer is organized as a tree, where subdirectories act as branches.


File and directory paths in UNIX use the forward slash "/" to separate directory names in a path.

Examples:

/	"root" directory
/usr	directory usr (sub-directory of / "root" directory)
/usr/d1	is a subdirectory of /usr
/tmp	directory to store temporary files
/home	directory with home folders for all registered users

MOVING AROUND THE FILE SYSTEM:

pwd	Show the "present working directory", or current directory.
cd	Change current directory to your HOME directory.
cd ~	Change current directory to your HOME directory.
cd /usr/IAC2016	Change current directory to /usr/IAC2016.
cd INIT	Change current directory to INIT which is a sub-directory of the current directory.
cd ..	Change current directory to the parent directory of the current directory.
cd ~jsmith	Change the current directory to the user jsmith's home directory (if you have permission).

 Exercise: navigate through your file system and list the contents of the following directories. Look-up/search for what is the purpose of each directory :

opt	_____	sbin	_____
bin	_____	usr	_____

dev	_____	root	_____
proc	_____	sys	_____
srv	_____	boot	_____
var	_____	home	_____
etc	_____	mnt	_____
lib	_____	run	_____
media	_____	tmp	_____

hints: <http://www.thegeekstuff.com/2010/09/linux-file-system-structure/>

CREATING AND REMOVING DIRECTORIES:

`mkdir dir1 [dir2...] create directories`

`mkdir -p dirpath` create the directory `dirpath`, including all implied directories in the path.

`rmdir dir1 [dir2...] remove an empty directory`

Exercise:

- | | |
|---|----------------------------|
| 1. Change the current directory to your home folder: | <code>cd</code> |
| 2. Create a new directory named IAC1617: | <code>mkdir IAC1617</code> |
| 3. Verify that your newly created directory has been created: | <code>ls</code> |
| 4. Change the current directory to the new folder: | <code>cd IAC1617</code> |
| 5. Create a new folder named Lab1 inside: | <code>mkdir Lab2</code> |
| 6. Because of the bad directory name, we need to delete it: | <code>rmdir Lab2</code> |
| 7. Return to your home directory using: | <code>cd ..</code> |
| 8. Confirm you're back in your home directory: | <code>pwd</code> |

MOVING, RENAMING, AND COPYING FILES:

A set of basic commands to help organizing the file system.

<code>cp file1 file2</code>	copy a file
<code>mv file1 newname</code>	move or rename a file
<code>mv file1 ~/IAC/</code>	move file1 into sub-directory IAC in your home directory.
<code>rm file1 [file2 ...]</code>	remove or delete a file
<code>rm -r dir1 [dir2...]</code>	recursively remove a directory and its contents BE CAREFUL!

Exercise:

- Copy the file from `/etc/hostname` to your new directory (this file contains the name of your computer): `cp /etc/hostname ~`
(or type `echo no_name > ~/hostname`)
- Verify the file has been correctly copied using `ls -l`. What's the file size?
- Rename the newly copied file `hostname` to `computer_name.txt`: `mv ho+TAB computer_name.txt`
- Copy the file `/etc/shells` to your home directory, but naming it `shells.txt`:

`cp /etc/shells shells.txt`

CHANGING FILE PERMISSIONS AND ATTRIBUTES

Files and folders can be accessed by all users or a set or a group of authorized users. Permissions can be changed using command `chmod`, `chgrp`.

As demonstrated before, you can check each files' permission using `ls -l`.

The attribute value encodes the permissions using 3 digits (0-7) or a character (r=read /w=write/x=execute) for user, group and everyone else (in this order).

`chmod 755 <file>` Changes the permissions of file to be rwx for the owner, and rx for the group and the world. (7 = rwx = 111 binary. 5 = r-x = 101 binary)

`chgrp user file` Makes file belong to the group user.

`chown ist1234 file` Makes ist1234 the owner of file.

`chown -R ist1234 dir` Makes ist1234 the owner of dir and everything in its directory tree.

You must be the owner of the file/directory or be root before you can do any of these things. The execute attribute is particularly important to make scripts executable.



Exercise:

Change the attributes of shells.txt to be read-write by the owner and the group and read-only by everyone else:

```
ls -l
chmod 664 shells.txt
ls -l
```

REDIRECTION:

It is used to change the input source from the keyboard to a file, and the output from the console to a file. If there are no decisions to be made by the user, this is particularly useful to automate your scripts.

`echo string > newfile` Redirects the output of the echo command to a file 'newfile'

`echo string >> existfile` Appends the output of the echo command to the end of 'existfile'

Using `<` you can redirect the input to come from a file rather than the keyboard.



Exercise: create a file with assorted numbers and sort it:

```
echo 7 > numlist
echo 3 >> numlist
echo 1 >> numlist
echo 4 >> numlist
sort < numlist
```

and the sorted list will be output to the screen.

```
ist14551@sigma01:~$ sort < numlist
1
3
4
7
```

To output the sorted list to a file, type:

```
sort < numlist > sorted_numlist
cat sorted_numlist
```

The redirection directives, `>` and `>>` can be used on the output of most commands to direct their output to a file. Command `cat` will show the file's contents:

```
ist14551@sigma01:~$ cat sorted_numlist
```

1
3
4
7

PIPES:

The pipe symbol "|" is used to direct the output of one command to the input of another, without using files.



Exercise:

Lists the contents of /etc directory one terminal screen at a time.

```
ls -l /etc | more
```

The `ls` command lists the contents of /etc directory and `more` displays one terminal screen at a time, waiting for the user input: SPACE continues to present the next screen and `q` exits the command.

This commands takes the output of the long format directory list command "`ls -l`" and pipes it through the `more` command (also known as a filter). In this case a very long list of files can be viewed a page at a time.

```
total 1412
drwxr-xr-x  3 root root    4096 Jul 27  2015 acpi
-rw-r--r--  1 root root    2986 Sep 14  2009 adduser.conf
-rw-r--r--  1 root root      45 Nov 25  2013 adjtime
drwxr-xr-x  2 root root   20480 Aug 17 13:30 alternatives
drwxr-xr-x  2 root root    4096 Jul 27  2015 anthy
drwxr-xr-x  3 root root    4096 Jul 27  2015 apache2
drwxr-xr-x  6 root root    4096 Jun  3 01:55 apt
```



Exercise:

Lists the largest files on the /etc directory

```
cd /etc
du -sc /etc/* | sort -n | tail
```

The command "`du -sc`" lists the sizes of all files and directories in the current working directory. That is piped through "`sort -n`" which orders the output from smallest to largest size. Finally, that output is piped through "`tail`" which displays only the last few (which just happen to be the largest) results.

```
ist14551@sigma01:~$ du -sc /etc/* | sort -n | tail
du: cannot read directory /etc/audisp: Permission denied
du: cannot read directory /etc/audit: Permission denied
du: cannot read directory /etc/lost+found: Permission denied
du: cannot read directory /etc/mysql/.oldstuff: Permission denied
du: cannot read directory /etc/polkit-1/localauthority: Permission denied
du: cannot read directory /etc/ssl/private: Permission denied
224    /etc/texmf
232    /etc/ImageMagick-6
296    /etc/ssh
336    /etc/sane.d
348    /etc/X11
352    /etc/init.d
632    /etc/joe
732    /etc/mono
1336   /etc/ssl
10680  total
```

ALIAS

Used to replace a frequently used long command with a shorter one. They will help you to execute commands faster. If you're using bash shell, it may be worth editing the .bashrc in your home directory to include your alias, so they'll be loaded automatically when the shell starts.

```
alias name='command'
```

Assign a name to a command, or set of commands.



Exercise: create an alias for “ls -la” command using only one character.

```
$ alias d='ls -la'

$ d
total 8
drwxrwxrwt+ 1 rpd None 0 Sep 12 15:44 .
drwxr-xr-x+ 1 rpd None 0 Sep 7 16:44 ..
drwxr-xr-x+ 1 rpd None 0 Sep 12 15:44 rpd
```

VIEWING AND EDITING FILES ON-SCREEN:

clear	Clears the terminal screen
cat filename	Dumps a file to the screen in ascii.
more filename	Progressively dumps a file to the screen: ENTER = one line down SPACEBAR = page down q=quit
less filename	Like more, but you can use Page-Up too. Not by default on all systems.
head filename	Shows the first few lines of a file
head -n filename	Shows the first n lines of a file.
tail filename	Shows the last few lines of a file.
tail -n filename	Shows the last n lines of a file.
vi filename	Edits a file using the vi editor. All UNIX systems will have vi in some form.
emacs filename	Edits a file using the emacs editor. Not all systems will have emacs.
file filename	Determine type of FILES

BASICS OF THE VI EDITOR

While working with vi editor at each moment, the editor will be in one of the following two modes:

Command mode – This mode enables you to perform administrative tasks such as saving files, executing commands, moving the cursor, cutting (yanking) and pasting lines or words, and finding and replacing. In this mode, whatever you type is interpreted as a command.

Insert mode – This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally it is put in the file .

vi always starts in command mode. To enter text, you must be in insert mode. To come in insert mode you simply type i. To get out of insert mode, press the **Esc** key, which will put you back into command mode.

Hint – If you are not sure which mode you are in, press the Esc key twice, and then you'll be in command mode.

OPENING A FILE

vi filename

CREATING TEXT

Edit modes: These keys enter editing modes and type in the text of your document.

- i Insert before current cursor position
- I Insert at beginning of current line
- a Insert (append) after current cursor position
- A Append to end of line
- r Replace 1 character
- R Replace mode
- <ESC> Terminate insertion or overwrite mode

DELETION OF TEXT

- x Delete single character
- dd Delete current line and put in buffer
- ndd Delete n lines (n is a number) and put them in buffer
- J Attaches the next line to the end of the current line (deletes carriage return).

OOPS

- u Undo last command

CUT AND PASTE

- yy Yank current line into buffer
- nyy Yank n lines into buffer
- p Put the contents of the buffer after the current line
- P Put the contents of the buffer before the current line

CURSOR POSITIONING

- ^d Page down
- ^u Page up
- :n Position cursor at line n
- :\$ Position cursor at end of file
- ^g Display current line number
- h,j,k,l Left, Down, Up, and Right respectively. Your arrow keys should also work if your keyboard mappings are anywhere near sane.
- 0 or | Positions cursor at beginning of line.
- \$ Positions cursor at end of line.
- w Positions cursor to the next word.
- b Positions cursor to previous word.
- (Positions cursor to beginning of current sentence.
-) Positions cursor to beginning of next sentence.
- E Move to the end of Blank delimited word
- { Move a paragraph back
- } Move a paragraph forward

STRING SUBSTITUTION

`:n1,n2:s/string1/string2/[g]` Substitute string2 for string1 on lines n1 to n2. If g is included (meaning global), all instances of string1 on each line are substituted. If g is not included, only the first instance per matching line is substituted.

`^` matches start of line

`.` matches any single character

`$` matches end of line

These and other "special characters" (like the forward slash) can be "escaped" with `\` i.e to match the string `"/usr/IAC/SOFT"` say `"\usr\IAC\SOFT"`

EXAMPLES:

`:1,$:s/dog/cat/g` Substitute 'cat' for 'dog', every instance for the entire file - lines 1 to \$ (end of file)

`:23,25:s/frog/bird/` Substitute 'bird' for 'frog' on lines 23 through 25. Only the first instance on each line is substituted.

SAVING AND QUITTING AND OTHER "EX" COMMANDS

These commands are all prefixed by pressing colon (`:`) and then entered in the lower left corner of the window. They are called "ex" commands because they are commands of the **ex** text editor - the precursor line editor to the screen editor vi. You cannot enter an "ex" command when you are in an edit mode (typing text onto the screen)

Press `<ESC>` to exit from an editing mode.

`:w` Write the current file.

`:w new.file` Write the file to the name 'new.file'.

`:w! existing.file` Overwrite an existing file with the file currently being edited.

`:wq` Write the file and quit.

`:x` idem.

`:q` Quit.

`:q!` Quit with no changes.

`:e filename` Open the file 'filename' for editing.

`:set number` Turns on line numbering

`:set nonumber` Turns off line numbering



Exercise:

```
cd
vi temp.txt
```

vi will create the file "temp.txt" in your home directory.

1. Press the "i" key to switch to input mode.
2. Type something like, "Vi is great! I think I'll be using vi from now on instead of Word"
3. Press `<ENTER>` to add lines.
4. Type some more text

Save the file that you are in. To do this press the ESCape key for command mode

Type “:wq” to save and quite the file (notice the “:” before the “wq”!).

Remember vi uses "modes". The easiest thing to do if you get confused in vi is to press the ESCape key a couple of times and start over with what you were doing.

Exercise:

Now, copy a large file to your home directory so that you can play around with some more vi commands.

You'll copy your /etc/defaults/rc.conf file to your home folder for this exercise. To do this do:

```
$ cd
$ cp /etc/sysctl.conf ~/sysctl.txt
```

Edit the file, but let's start at the bottom of the file:

```
$ vi + sysctl.txt
```

Go to the first line of the file. Notice the colon (":") before the "1".

```
:1
```

Go to line 10, add a new line, and add in some text:

```
:10
```

Press the "o" key

Add the following text:

```
##
## A sample comment
##
```

Delete the three lines you just created:

Move to the first line of new text. Press the ESCape key

Press "dd" to delete a line, repeat until the text is gone. Save the file, but don't exit.

```
:w
press <ENTER>
```

Practice copying and pasting text.

Go to line 10, copy 10 lines of text, go to the bottom of the file, and place the text there:

```
ESC
(go to command mode)
:12
(go to line 12 of the file)
3yy
```

("yank" 3 lines of text and place in copy buffer)

```
G
(go to the end of the file)
p
(place the contents of the copy buffer here)
```

If want to undo this you would type (in command mode):

```
u
```

Go to the top of the file, replace all occurrences of “mail” with “smtp”, but prompt for each change:

```
ESC  
:l  
:%s/ipv4/ipv6/gc
```

Say “yes” or “no” to a few prompts then escape from this mode by pressing ctrl-c and <ENTER>.

Go to line 1, search for “kernel”, move to the end of the line, and add some text:

```
ESC  
:l  
/kernel  
SHIFT-A  
"text here"  
ESC
```

Now let’s exit from the file and save the few changes we’ve made.

```
:x
```

Or


```
:wq
```

Or

```
:w  
:q
```

To exit without saving write:

```
:q!
```

 There are many webpages dedicated to vi, and other smart-text-editors such as emacs. Their usage is not straightforward, but it’s worth investing some time trying and learning to use them.

PROCESSES AND JOBS

A process is an executing program identified by a unique PID (process identifier). To see information about your processes, with their associated PID and status, type

```
ps
```

A process may be in the foreground, in the background, or be suspended. In general the shell does not return the UNIX prompt until the current process has finished executing.

Some processes take a long time to run and hold up the terminal. Backgrounding a long process has the effect that the UNIX prompt is returned immediately, and other tasks can be carried out while the original process continues executing.

RUNNING BACKGROUND PROCESSES

To background a process, type an **&** at the end of the command line. For example, the command sleep waits a given number of seconds before continuing. Type

```
sleep 10
```

This will wait 10 seconds before returning the command prompt %. Until the command prompt is returned, you can do nothing except wait.

To run sleep in the background, type

```
% sleep 10 &  
[1] 6259
```

The **&** runs the job in the background and returns the prompt straight away, allowing you to run other programs while waiting for that one to finish.

The first line in the above example is typed in by the user; the next line, indicating job number and PID, is returned by the machine. The user is notified of a job number (numbered from 1) enclosed in square brackets, together with a PID and is notified when a background process is finished. Backgrounding is useful for jobs which will take a long time to complete.

BACKGROUNDING A CURRENT FOREGROUND PROCESS

At the prompt, type:

```
sleep 1000
```

You can suspend the process running in the foreground by typing **^Z**, i.e. hold down the **[Ctrl]** key and type **[z]**. (You can also stop by pressing **^C**) Then to put it in the background, type

```
bg
```

LISTING SUSPENDED AND BACKGROUND PROCESSES

When a process is running, backgrounded or suspended, it will be entered onto a list along with a job number. To examine this list, type:

```
jobs
```

An example of a job list could be:

```
[1] Suspended sleep 1000  
[2] Running netscape  
[3] Running matlab
```

Example:

```
ist14551@sigma01:~$ sleep 1000 &  
[1] 573  
ist14551@sigma01:~$ jobs  
[1]+  Running                  sleep 1000 &
```

To restart (foreground) a suspended process, type

```
fg %jobnumber
```

For example, to restart sleep 1000, type:

```
fg %1
```


Typing **fg** with no job number foregrounds the last suspended process.

You can suspend a process pressing **CTRL-Z** while it is running.

PS (PROCESS STATUS)

Alternatively, processes can be killed by finding their process numbers (PIDs) and using `kill PID_number`


```
sleep 1000 &
ps
```

 Exercise: List the processes currently running on your machine:

```
PID TT S TIME COMMAND
20077 pts/5 S 0:05 sleep 1000
21563 pts/5 T 0:00 netscape
21873 pts/5 S 0:25 nedit
```

To kill off the process **sleep 1000**, type

```
kill 20077
```

and then type **ps** again to see if it has been removed from the list.

If a process refuses to be killed, uses the **-9** option, i.e. type

```
kill -9 20077
```

Note: It is not possible to kill off other users' processes!

KILLING A PROCESS

KILL (TERMINATE OR SIGNAL A PROCESS)


It is sometimes necessary to kill a process (for example, when an executing program is in an infinite loop)

To kill a job running in the foreground, type **^C** (control c). For example, run

```
sleep 100
^C
```

To kill a suspended or background process, type

```
kill %jobnumber
```

 Exercise: Run **sleep** for 1000 seconds in background, identify it's job number and kill it.

```
sleep 1000 &
jobs
```

If it is job number 4, type

```
kill %4
```

To check whether this has worked, examine the job list again to see if the process has been removed.


OTHER USEFUL COMMANDS

DIFF

Diff tells you the differences between files.

```
% diff file1 file2          (basic use of diff)
% diff -i file1 file2       (tell diff to ignore case)
```

Try options: **-c**, **-e**, **-f**, **-h**, **-n**

 Exercise: Identify the differences between two text files

1. Create a text file with the information about the CPU(s) on your computer

```
lscpu > my_cpu.txt
```

2. Make a copy of that file, edit the new file and change the CPU speed for twice the current value and delete the first line.

```
cp my_cpu.txt new_cpu.txt
vi new_cpu.txt
```

3. Compare the two files

```
diff my_cpu.txt new_cpu.txt
```

You should get the identification of the lines on the new file that differ from the original one.

```
ist14551@sigma01:~$ diff my_cpu.txt new_cpu.txt
1d0
< Architecture:          x86_64
13c12
< Model name:           Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
---
> Model name:           Intel(R) Xeon(R) CPU E5-2620 0 @ 4.00GHz
```

Try to figure out what is the meaning of the numbers and letters before each line and the comparison signs (1d0, 13c12, >, <)

Try:

```
diff new_cpu.txt my_cpu.txt
```

Compare with the previous execution.

ZIP, UNZIP

zip reduces the size of the given file while unzip reverse the effect and expands the compressed file back to its original form. The default extension .zip is used for compressed files. There are other alternatives such as gzip and compress. Here's an example on how commands compress and uncompress are used.

Zcat is equivalent to 'uncompress -c'

```
% compress myfile          (myfile becomes myfile.Z)
% uncompress myfile.Z      (myfile.Z becomes myfile)
% uncompress -c myfile.Z   (dump content of myfile.Z to standard output,
                           myfile.Z does NOT become myfile)
% zcat myfile.Z            (same as above)
% zcat myfile.Z > myfile   (redirect stdout to a file)
```

Investigate other compression utilities such as gzip, and tar.



Exercise: Compress the largest file in /boot to your home directory:

Determine the largest file in /boot:

```
du -sc /boot/* | sort -n | tail -n 2
```

Compress it to your home folder

```
ist14551@sigma01:~$ zip boot.zip /boot/initrd.img-4.2.0-1-grsec-amd64
  adding: boot/initrd.img-4.2.0-1-grsec-amd64 (deflated 0%)
ist14551@sigma01:~$ ls -la boot.zip
-rw-r--r-- 1 ist14551 14551 17961758 Sep 23 14:06 boot.zip
```

Decompress it to your current folder (bear in mind that the directory will be reconstructed in your local folder.

```
ist14551@sigma01:~$ unzip boot.zip
Archive:  boot.zip
  inflating: boot/initrd.img-4.2.0-1-grsec-amd64
```

Now, let's compare the two files to see if they match. You're going to do this using MD5. MD5 is a checksum computed from a file's contents (think of it like the sum of bytes in the file), which is different for any two files even though they only differ in one bit!

Usage: md5sum file1 file2

```
ist14551@sigma01:~/boot$ md5sum  initrd.img-4.2.0-1-grsec-amd64  /boot/initrd.img-4.2.0-1-grsec-amd64
0b86d34e6b2fb2c6804fe64277756d6b  initrd.img-4.2.0-1-grsec-amd64
0b86d34e6b2fb2c6804fe64277756d6b  /boot/initrd.img-4.2.0-1-grsec-amd64
```

Since the checksum is the same for both, the two files are identical.

SEARCHING FOR STRINGS IN FILES

To search for strings in files there's a command named `grep`, and is used as follows:

```
grep string filename
```

It prints all the lines in a file that contain the string.

`grep` is one of many standard UNIX utilities. It searches files for specified words or patterns. The `grep` command is case sensitive; it distinguishes between *Science* and *science*.

To ignore upper/lower case distinctions, use the `-i` option, i.e. type

```
grep -i cpu new_cpu.txt
```

To search for a phrase or pattern, you must enclose it in single quotes (the apostrophe symbol). For example to search for spinning top, type:

```
grep -i 'cpu mhz' new_cpu.txt
```

Some of the other options of `grep` are:

- `-v` display those lines that do NOT match
- `-n` precede each matching line with the line number
- `-c` print only the total count of matched lines

✂ Exercise: Try some of them and see the different results. Don't forget, you can use more than one option at a time. For example, the number of lines without the words `cpu`, `Cpu` or `CPU` is

```
grep -ivc cpu science.txt
```


SEARCHING FOR FILES : THE *FIND* COMMAND

```
find search_path -name filename
```

`find . -name cpu.txt` Finds all the files named `cpu.txt` in the current directory or any subdirectory tree.

`find / -name iac` Find all the files named `iac` anywhere on the system.

`find /usr/local/games -name "*pacman*" Find all files whose names contain the string 'pacman' which exist within the '/usr/local/games' directory tree.`

 Exercise: Find all files named passwd inside /etc directory:

```
find /etc -name passwd
```

COMMAND SUBSTITUTION

You can use the output of one command as an input to another command in another way called command substitution. Command substitution is invoked when by enclosing the substituted command in backwards single quotes. For example:

```
cat `find . -name my_cpu.txt`
```

which will cat (dump to the screen) all the files named my_cpu.txt that exist in the current directory or in any subdirectory tree.

TUTORIAL SOURCES

1. <http://ryanstutorials.net/linuxtutorial/>
2. <http://freeengineer.org/learnUNIXin10minutes.html>
3. <http://www.cs.toronto.edu/~maclean/csc209/unixtools.html>
4. <http://www.doc.ic.ac.uk/~wjk/UnixIntro/>