Alien's Bash Tutorial

Written by Billy Wideling <-> alien a koping d net

*Modifications with kind permission from Billy Wideling: 2006-07-17 AL Original .txt file version from http://www.usc.edu/~lhl/tutorials/unix/bash_tutor/aliens_bash_tutorial.txt (now deleted), copied here. Converted to HTML, inserted a clickable Table of Contents and clickable Index of *nix commands, local 'targets', spell checked, fixed a lot of grammar and tabled some pictures.*
2008-08-17 AL Used div, pre and tt styles.
*2010-01-30 AL Used dt and dd styles. Fixed some broken indentation. Moved the list of *nix commands to the end of the document, since it's really an appendix. Added descriptive subsections and respective hyperlinks to the Table of Contents.*
2010-03-18 PH Changed coloring and mirrored it on http://subsignal.org
AL is Alf Lacis, alfredo4570 a gmail d com.
PH is phaidros aka kloschi, phaidros a subsignal d org.

First you probably need to read a UNIX command bible to really understand this tutorial, but I will try to make it as clear as possible, there is about 100-150 UNIX commands explained later in this tutorial.

You are to have some UNIX experience before starting on this tutorial, so if you feel that you have UNIX/Linux experience feel free to start to learn here.

What I included here is general shell scripting, most common other things and some UNIX commands.

Here's the most common shell types:

```
bash  = Bourne again shell
sh    = shell
csh   = C shell
tcsh  = Tenex C shell (not tab-completion-extended C shell)
tclsh = Tcl shell
ksh   = korn shell
ash   = a shell
bsh   = bourne shell? (in most Linux distributions it's a link to /bin/ash)
zsh   = the Z shell (it's what its manual page tells about it.. :/ )
```

Table Of Contents

1 - What you already know (should know)
- --> Shell command, operator, separator & control characters
- --> Basic UNIX or Linux directory structure
- --> File location examples
- --> Some basic introductory commands

2 - Where to start
- --> First examples

3 - Beginning techniques
- --> Calculator
- --> String comparison
- --> Indentation
- --> More on comparing strings
- --> Handling empty strings in comparisons
- --> Introducing operators
- --> Loops: `for` and `while` (with `case` example)
- --> Functions

4 - Other techniques
- --> Embedding C code in bash scripts
- --> More on the `case` command
- --> Quick look at `sed`
- --> The `dialog` box
- --> Checking for existence of a directory: "`[ -d ...`" or "`test -d ...`"
- --> Creating "lock" files
- --> Creating unique file names using `$$`
- Using a `$` variable to hold:
  - --> a command
  - --> the contents of a file
  - --> one line from a file
- --> Using arrays in shell scripting
- --> Using CGI & HTML in bash scripts

5 - Practical Scripting Examples

6 - Customize your system and shell
- --> Using other shells for a login
- --> `bash` environment variables
- --> Adding useful information to your command prompt
- --> Working with colors
- --> Aliases
- --> Scheduling repeat tasks with `cron`
- --> Mounting disks - definition file: `/etc/fstab`
- --> Global profile settings definition file: `/etc/profile` & the `$PATH` variable
- --> `/etc/hosts.allow` & `/etc/hosts.deny` files
- --> Keyboard configuration: `/etc/inputrc`
- --> Login information: `/etc/passwd` & `/etc/shadow`
- --> Message-of-the-day file: `/etc/motd`
- --> Basic 'skeleton' files for setting up new accounts: `/etc/skel/` directory
- --> `/etc/issue` & `/etc/issue.net`

7 - Networking
- --> Basic setup
- --> IP address, network name and nickname
- --> Forwarding & `ipchains`
- --> Caching nameserver

8 - The init and system scripts
- --> Init files and directories
- --> Runlevels
- --> BSD `init` and System V `init` differences
- --> Example `init` script
- --> The `/etc/inittab` file & the `/etc/rc.d/` directory structure

9 - Other Frequently asked questions with answers
- --> ... including using hex codes
- --> ... and an answer on long integers ("`DWORD`s")

10 - Basics of the common UNIX and Linux text editors

--> A note on paths: "./", "../", etc
--> Searching through users' histories, e.g., for "passwd" - caution
--> Wingate scanner - caution
--> dialog and Xdialog example to run mpg123
--> vi
--> ed
--> emacs
11 - Closing
Appendix A - Annotated Basic Linux/UNIX commands
--> Index of Annotated Commands

==

1 - What you already know (should know)

== Table Of Contents

Here we go, bash scripting is nothing more than combining lots of UNIX commands to do things for you, you can even make simple games in bash (just UNIX commands) or as in normal cases, batch files to control things in your computer.

There is a variety of easier and harder examples in the beginning of this tutorial, I've done it this way to make it easier for people to get the general picture, so they will get more of the "aha!" experiences in the later chapters of this tutorial.

What bash or any scripting language does is to call for premade programs that lives in your computer. So when you write a command in a script you are calling a command that is a part of the system. That is why this tutorial will be just as much a Linux and UNIX tutorial as a shell scripting tutorial.

I will however not take up much about The X Windows System in this tutorial, for the simple reason that a Window Manager does nothing else than display programs.

This means that a Window Manager is like a graphical shell for the system. You can do all in this tutorial from any terminal emulator in a Linux Graphical Environment (The X Windows System).

A terminal emulator would be such as: `Eterm`, `xterm`, `axvt`, `rxvt`, `kterm`, etc. A terminal emulator lets you get up a terminal with a command prompt in a graphical window.

Shell command, operator and separator/control characters:

```
|  = pipe will take the first commands stdout as the second commands stdin.
|| = OR if first command is false, it will take the second.
|= = OR IS (mostly used in if statements)
&& = AND if first command is true, it will execute the second one.
!  = NOT (mostly used in if and test statements), but as a shell-command
     it opens a shell to run the command (ex. `! echo foo`)
!= = NOT IS (mostly used in if statements)
!$ = last commands last argument
!! = repeat last command
=  = IS (mostly used in if statements)
;  = will separate 2 commands as if they were written on separate command lines
;; = end of a case function in a case statement. (see 'case' further down)
$  = prefix to a variable like "$myvar"
$! = PID of the last child process.
$$ = PID of current process (PID == Process ID)
$0 = Shows program that owns the current process.
$1 = First argument supplied after the program/function on execution.
$2 = Second argument supplied after the program/function on execution. ($3 etc.)
$# = Shows the number of arguments.
$? = Any argument (good to use in 'if' statements)
$- = current option flags (I never ever had to use this one)
$_ = Last argument/Command
$* = All arguments
$@ = All arguments
#  = remmed line, anything on a line after "#" will be overlooked by the script
{  = start braces (starts a function)
}  = end braces   (ends a function)
[  = start bracket (multiple-argument specifiers)
]  = end bracket (multiple-argument specifiers)
@  = $@ is equivalent to "$1" "$2" etc. (all arguments)
*  = wild card (* can substitute any number of characters)
?  = wild card (? can substitute any single character)
"  = quote
'  = precise quote. (Will even include "'s in the quote)
`  = command quote. (variable=`ls -la` doing $variable will show the dir list)
.  = dot will read and execute commands from a file, ( . .bashrc )
&  = and. as suffix to executed file makes it go to the background(./program &)
0> = stdin stream director (I never seen this used in any script)
1> = stdout stream director (standard output)
2> = stderr stream director (standard error output)
%  = job character, %1 = fg job 1, %2 = fg job 2, etc.
```

```
>> = stream director append to a file
<< = stdin stream director. (cat > file << EOF ; anything ; EOF)
>  = stream director that will start at the top of the file
    (in if statements < and > may be used as greater-than
    and lesser-than, as: if [ "$1" >= "2" ])
\  = back-slash, takes away any special meaning with a character,
     \$var will not be treated as a variable.
     (and a new line will not be treated as a new line)
     Also a \ before a command, removes any alias on the command as: \rm
>& = stream director to stream director, ie. echo "a" 1>/dev/null 2>&1
     this directs 2> to the same place as 1>
```

Here is the basic UNIX or rather Linux directory structure:

```
                                / (system root)
 _______________________________|____________________________________
 |    |     |    |     |    |       |         |    |      |    |    |
bin/ boot/ dev/ etc/ home/ lib/ lost+found/ proc/ root/ sbin/ usr/ var/
 |    |     |    |     |    |       |         |    |      |    |    |
 |    |     |    |     |    |       |         |    |      |    |    |-> various
 |    |     |    |     |    |       |         |    |      |    |
 |    |     |    |     |    |       |         |    |      |    |-> Read later..
 |    |     |    |     |    |       |         |    |      |
 |    |     |    |     |    |       |         |    |      |-> Superuser Binaries
 |    |     |    |     |    |       |         |    |
 |    |     |    |     |    |       |         |    |-> User roots home dir
 |    |     |    |     |    |       |         |
 |    |     |    |     |    |       |         |-> system info etc.
 |    |     |    |     |    |       |
 |    |     |    |     |    |       |-> Orphan files (look here after crashes)
 |    |     |    |     |    |
 |    |     |    |     |    |-> System / programming libraries
 |    |     |    |     |
 |    |     |    |     |-> Contains the user accounts home directories
 |    |     |    |
 |    |     |    |-> System configuration files.
 |    |     |
 |    |     |-> Devices
 |    |
 |    |-> The kernel and kernel maps.
 |
 |-> Executable files. (binaries)
```

The `/usr` directory contains a whole lot of things, mainly user accessible things, like binaries in `/usr/local/bin/` and `/usr/bin/` also libraries in `/usr/lib/` and `/usr/local/lib/`.

The kernel source should also be under `/usr`, in `/usr/src/linux/` But more about that later.

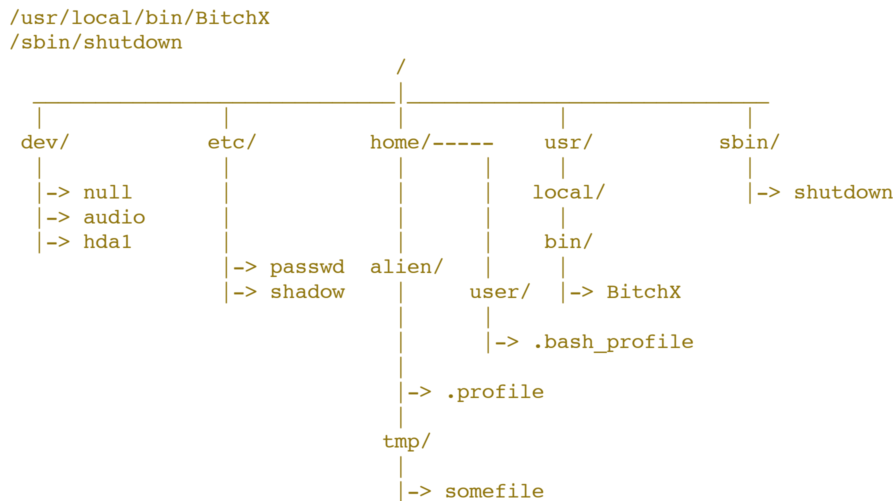
Here's an example of the following files locations:

```
/dev/null
/dev/fd0
/etc/passwd
/home/alien/.profile
/usr/local/bin/BitchX
                                /
 _______________________________|____________________________________
 |    |     |    |     |    |       |         |    |      |    |    |
bin/ boot/ dev/ etc/ home/ lib/ lost+found/ proc/ root/ sbin/ usr/ var/
            |    |     |                                       |
            |    |   alien/                                  local/
            |    |     |                                       |
            |    |     |-> .profile                            |
            |    |-> passwd                                   bin/
            |                                                  |
            |-> null                                           |-> BitchX
            |-> fd0
```

And a final example that is a bit more stripped, just in case you are really new to computers. For these following files, the locations are:

```
/dev/null
/dev/audio
/dev/hda1
/etc/passwd
/etc/shadow
/home/alien/.profile
/home/alien/tmp/somefile
/home/user/.bash_profile
```

```
/usr/local/bin/BitchX
/sbin/shutdown
```



This is the same structure as on any Operating System that uses directories, Though some Operating Systems may call the `/` directory `C:` and some other may call it `HD` etc. and of course some of the directory names in Linux/UNIX are UNIX specific.

No further explanation should be necessary.

After reading this tutorial, browse around the system and try to learn what all the files does, just don't remove any files you didn't put there until you're ABSOLUTELY sure of what you are doing.

Here's a few UNIX commands just for illustration:

`echo`

`echo` will *echo* anything you add to it like this:

```
alien:~$ echo "blah"
blah
alien:~$
```

To get it to echo without a new line add the suffix -n like this:

```
alien:~$ echo -n "blah "
blah
alien:~$
```

I'll get back to why you want to do "-n" sometimes in a while.

`read`

`read` will read from the keyboard (stdin) and save it as a variable. The variable name goes after the `read` command, like this:

```
alien:~$ read myvar
<here I type say "blah">
alien:~$ echo $myvar
blah
alien:~$
```

To combine these two commands (`echo` and `read`) in a small script line, it can look like this:

```
alien:~$ echo -n "password: " ; read pass ; echo "Your pass is $pass"
password: <here I type "mypass">
Your pass is mypass
alien:~$
```

Get the basic idea?

Anyway, here are some commands that you should know before moving on with this tutorial -

```
[*] after == important to know.
[X] after == very basics will do.

ls      [*]     Ex: ls -la              Long directory listing.
echo    [*]     Ex: echo "foo"          Does what it says.
cat     [*]     Ex: cat /etc/passwd     Dump out the content of a file.
less    [X]     Ex: less /etc/passwd    Scroll up and down in a file (q = exit)
head    [X]     Ex: head -5 /etc/passwd Get the 5 (-5) first lines of a file.
tail    [X]     Ex: tail -7 /etc/passwd Get the 7 (-7) last lines of a file.
grep    [*]     Ex: grep x /etc/issue   Dump lines containing x from /etc/issue
chmod   [*]     Ex: chmod a+x file      Give everyone executable rights to file
chown   [X]     Ex: chown root file     Change owner of file to root.

(cd     [-]     Ex: cd /etc             Change Directory to /etc)
```

Some applications you need to know how to operate (basics will do) :

```
any text editor (preferably emacs or vi - they are explained last in this file)
telnet          Ex: telnet 127.0.0.1    Opens a connection to IP 127.0.0.1
lynx            Ex: lynx http://foo.bar A command line based web browser.
ftp (ncftp)     Ex: ncftp ftp://foo.bar A command line based ftp client.
ssh             Ex: ssh 127.0.0.1       Opens a secure connection to 127.0.0.1
```

These are all explained fully later in this tutorial.

Now don't sit there and ask yourself how's going to teach you the commands or applications I just listed here above. use the manual pages. like this:

```
man echo
```

that will get you the full manual on the command echo :) `man` works the same way with applications that you have the manual pages for.

To get out of the manual page just press the letter `q`. `q` quits it and gets you back to the command line prompt. `man` uses all the normal `less` commands.

Or read further down in this tutorial in the basic Linux/UNIX commands and operations section (9).

The key to shell scripting just as with any programming language/Operating System is to REALLY understand what you are doing, so do read this file more than once, and don't read it too fast.
Take your time and let it sink in, so you know what it's all about, and do take time to read manual pages and do some playing with the commands so you learn them.

Now that should be enough of what you *should* know before starting to learn UNIX shell scripting.

So here we go.......

```
===============================================================================
```

2 - Where to start

== Table Of Contents

You should always start with very simple scripts that you really don't have any practical use for but still *could* be of practical use =)

As for first let's make what we already know to a *real* executable script. Open a text editor (name the file `myscript.sh`) and type this:

```
#!/bin/bash

echo -n "password: "
read pass
echo "Your pass is $pass"
```

save & exit --- then do this command:

```
chmod u+x myscript.sh
```

Then we can execute it:

```
alien:~$ ./myscript.sh
password: <type what you want>
Your pass is <what you typed>
alien:~$
```

The "`#!/bin/bash`" at the start of the file is to let the script know what shell type it should use.

The "`chmod u+x myscript.sh`" is to make it executable.
(English: change-mode user+execute-right-on myscript.sh) ..... read the manual pages on `chmod` for more info on it =)

Take a lot of time to play around in your system, open files, figure out what they do (but don't edit or remove them).

Take time also to learn some good text editor, that is important. Learn, `emacs` or `vi`, those are by most people considered the absolutely best, but `jed`, `joe`, `elvis`, `pico` or any simple editor like that will do just fine for now.

`emacs` and `vi` are explained later in this tutorial.

Another thing before moving on is that you can export a variable from a script.

Say that you have the variable `$var` in a script and want to export it to the system for use with some other script or something, you can do: `export var` .

Like this little script:

```
#!/bin/bash

VAR="10"
export VAR
```

Note: `VAR="10"` can **not** be written as `VAR = "10"`, because it's 'whitespace-sensitive'.

But more to how to make scripts in a second, I just thought that this would be a good time to enlighten you about this.

So here we go ....

==

3 - Beginning techniques

== Table Of Contents

First off I'm going to show how to count in bash or how to use your command line as a calculator, which is really easy and useful.

```
alien:~$ echo $[ 4 * 2 ]
8
alien:~$
```

or another example:

```
alien:~$ echo $[ 10 + 5 ]
15
alien:~$
```

Easy? .... I think it is =)

The internal calculator can also be used like this:

```
alien:~$ echo $(( 10 + 5 ))
15
alien:~$
```

The second way of using the internal shell calculator is here just so you don't get confused if you see it used that way sometime.

Now I'd like to show the *string comparing* with "`if`" statements, which can be a little hard at first contact, but as soon as you get familiar with it, it won't be any problem. So here's an example.

```
#!/bin/bash

echo -n "enter a name: "
read var1
echo -n "enter another name: "
read var2

if [ "$var1" = "$var2" ]; then
    echo "The names are the same"
else
    echo "The names were not the same"
fi

exit 0
```

Note: `fi` is the ending of `if`, just like a `}` is the ending of a `{`.
"`exit 0`" terminates the script correctly and returns you to the prompt.

Another note is that instead of " `=` " you can use " `-eq` " to test if 2 expressions are equal, or " `-eg` " to check if 2 integers are equal, etc.

Indentation:
AL: Notice how the parts of the script between the `then`, the `else` and the `fi` are indented? These help you keep track visually what will happen in the script, but are not strictly required.
Indenting becomes important on really larger scripts, so just learn to do it all the time after an `if`, `while`, etc, and it will become natural.

It should also be said that a variable say "`$var`" can be written as this: `${var}`, just so you know if you see it done that way in some scripts, but here we will use the `$var` way.

AL: `${var}` is especially useful where parts of a variable name have to be joined up, e.g., if `$foo` contains `Hello` then `var=${foo}XYX` will set `$var` to `HelloXYZ`.

This example if executed looks like this:
(Matching names)

```
alien:~$ ./script1.sh
enter a name: smurf
enter another name: smurf
The names are the same
alien:~$
```

(Non-matching names)

```
alien:~$ ./script1.sh
enter a name: smurf
enter another name: guru
The names were not the same
alien:~$
```

You can compare any 2 strings with this, as this *mid script example*:

```
...
if [ "$user" = "gnu" ]; then
   echo "Hello user gnu !"
else
   echo "I don't know you."
fi
...
```

This compares a variable with a static string which you can set to anything. You can also do this the other way around.

```
...
if [ "$user" != "gnu" ]; then
   echo "I don't know you."
else
   echo "Hello user gnu !"
fi
...
```

The "`!=`" means NOT-IS, in clear text if the 2 strings are not a match. As the above example in straight English:

```
...
if the variable doesn't match the word gnu, then
    say "I don't know you."
in other cases
    say "Hello user gnu !"
...
```

If you think that a variable may not contain anything and you wanna avoid it showing you errors you can add an `x` (or any other character) as the first character to both the statements to test with if, like this to compare `$one` with `-x`:

```
...
if [ "x$one" = "x-x" ]; then
    echo "$one is -x"
else
    echo "$one is not -x"
fi
...
```

In plain English:

```
...
if (contents of $one) equals -x (suppress error messages if any), then
    say (contents of $one) is -x
in other cases
    say (contents of $one) is not -x
...
```

This previous way is actually quite old, and only a precaution, say this:

```
...
echo -n "enter a number: "
read foo
if [ $foo = 2 ]; then
    echo ok foo
fi
...
```

Now, if you with this example don't enter any number there will be nothing there for if to compare with, not even a blank "", since we're not using quotes, but as this:

```
...
echo -n "enter a number: "
read foo
if [ x$foo = x2 ]; then
    echo ok foo
fi
...
```

There will always be something, because if `$foo` is nothing there is still `x`. Just read a couple of times and you'll get it.

You can also test if a variable contains anything at all like this:

```
...
echo -n "enter a number: "
read foo
[ -n $foo ] &&
    echo ok foo
...
```

This uses the same options as the test command, so "`-z`" will return true if the variable is empty, `-z` will return true if the variable is not empty etc. It's ok if you don't understand this right now .... I've added this for the second time readers.

You can also test if a command turns out as true, like this:

```
...
if echo "foo" >/dev/null; then
    echo "foo"
else
    echo "bar"
fi
...
```

Here if will check if `foo` echos to `/dev/null`, and if so, then it will print out `foo` and if `foo` didn't echo to `/dev/null`, it'll print out the word `bar`.

Another and perhaps *cleaner* way of doing the same is this:

```
...
if (echo "foo" >/dev/null); then
    echo "foo"
else
    echo "bar"
fi
...
```

It's the exact same thing but with parentheses around the command, it looks much cleaner ... and so the code is easier to follow.

You can also make if think 'if this is a match or that is a match', like if the variable is one of two options do one thing else do another. Like this:

```
...
if [ "$user" = "gnu" -o "$user" = "ung" ]; then
    echo "Hello $user !"
```

```
else
    echo "I never heard of $user..."
fi
...
```

The "`-o`" means OR in an `if` statement, so here is the example in plain English:

```
...
if the variable matches the word gnu or matches the word ung, then
    say "Hello word !" (the word is the variable, now gnu or ung)
in other cases
    say "I never heard of word..." (the word is whatever the variable is set to)
...
```

Note: The quotes are needed in an `if` statement in case the strings or variables it's suppose to compare are empty, since

```
if [ foo =  ]; then
```

would produce a syntax error, but

```
if [ "foo" = "" ]; then
```

would not.

The `-o` can also be made with "`] || [`", so that:

```
if [ "$user" = "gnu" -o "$user" = "ung" ]; then
```

can also be expressed as this:

```
if [ "$user" = "gnu" ] || [ "$user" = "ung" ]; then
```

You don't really need to remember that, but for the knowledge I decided to make a note out of that anyway, mostly for the more experienced readers of this tutorial, and for the readers that have read it several times.

You can also set static text in a variable, which is really easy:

```
#!/bin/bash
anyvar="hello world"
echo "$anyvar"
exit 0
```

Which executed would look like this:

```
alien:~$ ./myscript
hello world
alien:~$
```

Easy enough? =)

Loops

Now let's move on to "`for`" and common `for` loops.

I am actually only going to show one sort of `for` loop example, of the reason that at this stage no more is needed, and would only confuse. As a note, `for` loops can be used (as soon shown) to import strings from a file to be used as variables in the script.

Now, here's the example:

```
#!/bin/bash
for VAR in `cat list.txt`; do
    echo "$VAR was found in list.txt"
done
exit 0
```

Note: "`done`" terminates the loop when finished.
    "`in`" and "`do`" are like bash *grammar*, I'll explain that later.

In the `` `cat list.txt` `` part, the `s around the command will make sure the script/line executes that part as a command, another way of doing this is to: `$(cat list.txt)` which has the same effect.
That is just a note so you won't get confused if you see it used that way some time.

The previous script example is dependent on that there is a file called `list.txt`, so let's make such, and fill it with something like this:

```
$ cat list.txt
123 234 345
456 567 678
789 890
```

Then the executed script would look like this:

```
alien:~$ ./script2.sh
123 was found in list.txt
234 was found in list.txt
345 was found in list.txt
456 was found in list.txt
567 was found in list.txt
678 was found in list.txt
789 was found in list.txt
890 was found in list.txt
alien:~$
```

Note: A space in a file read by a `for` loop is taken the same way as a new line.

Here is another example, with a `for` loop with an if statement:

```
#!/bin/bash

for VAR3 in `cat list.txt`; do
    if [ "$VAR3" = "789" ]; then
        echo
        echo "Match was found ($VAR)"
        echo
    fi
done

exit 0
```

And executed that looks like this:

```
alien:~$ ./script3.sh

Match was found (789)

alien:~$
```

If you have read this in a calm fashion it should be quite clear to you so far, but before I move on to real practice examples I will explain the `while` loop, and some, more which can be used as to count and more, for various purposes, as you will see. You don't have to *understand* all of how this works, but you should at least learn it.

So here we go on an example with `while`:

```
#!/bin/bash

count="0"
max="10"

while [ $count != $max ]; do count=`expr $count + 1`
    echo "We are now at number: $count"
done

exit 0
```

Note: `expr` is a calculator command, you can read more about it later in this tutorial.

This in plain English reads the following:

```
make variable "count" hold the number 0
make variable "max" hold the number 10

while 0 is not 10, do add 1 to 0 (each loop until it is 10)
```

```
    say "We are now at number: $count" (each time 1 is added as long as we are
    in the loop)
end the loop
return to the prompt command line.
```

Which executed looks like, (you guessed it), this:

```
alien:~$ ./count.sh
We are now at number: 1
We are now at number: 2
We are now at number: 3
We are now at number: 4
We are now at number: 5
We are now at number: 6
We are now at number: 7
We are now at number: 8
We are now at number: 9
We are now at number: 10
alien:~$
```

Here is another example of a while loop.

```
#!/bin/bash

agreement=
while [ x$agreement = x ]; do
    echo
    echo -n "Do you agree with this? [yes or no]: "
    read yesnoanswer
    case $yesnoanswer in
        y* | Y*)
                agreement=1
                ;;
        n* | n*)
                echo "If you don't agree, you can't install this software";
                echo
                exit 1
                ;;
    esac
done

echo "agreed"
echo
```

This in plain English reads the following:

```
Make an unknown variable named agreement
while the unknown variable is unknown and doesn't match the case,
    say "Do you agree with this? [yes or no]: "
    read the answer into the "yesnoanswer" variable.
    make a case and check the "yesnoanswer" variable for any words beginning
        with y or Y, and if so, skip the rest and go on with the script
        and say "agreed".
    if it doesn't begin with y or Y, check if it starts with n or N.
        If it does start with a n or N, then say:
        "If you don't agree, you can't install this software"
and quit the script.
```

Which executed looks like this:

```
alien:~$ ./agree.sh

Do you agree with this? [yes or no]: something

Do you agree with this? [yes or no]: yes
agreed
```

Executed again, but with no as the answer:

```
alien:~$ ./agree.sh

Do you agree with this? [yes or no]: nothing
If you don't agree, you can't install this software

alien:~$
```

Note that "`nothing`" begins with `n` and therefore matches what the script is looking for, `y` or `Y`, and `n` or `N`.

Also see later in the tutorial about `case` statements.

Functions

Now I'm going to explain shortly about functions in bash.

A function is like a script within the script, or you could say that you make your own little command that can be used in a script. It's not as hard as it sounds though.

So here we go on a example:

```
#!/bin/bash

function myfunk {
    echo
    echo "hello"
    echo "this is my function"
    echo "which I will display twice"
}

myfunk
myfunk

exit 0
```

Which executed looks like this:

```
alien:~$ ./funk.sh

hello
this is my function
which I will display twice

hello
this is my function
which I will display twice
alien:~$
```

Another example of functions can look like this:

```
#!/bin/bash

myvar="$1"
var2="$2"

if [ "$myvar" = "" ]; then
    echo "Usage: $0 <integer> <integer>"
    exit 0
fi

function myfunk {
    expr $1 + $2
}

myfunk $myvar $var2

exit 0
```

Which executed looks like this:
Without any arguments:

```
alien:~$ ./funk.sh
Usage: ./funk.sh <integer> <integer>
```

With arguments:

```
alien:~$ ./funk.sh 12 3
15
alien:~$
```

Note: the `$1` and `$2` in the function is in fact the first and second argument supplied after the function name when it's called for within the

script, so you could say that a function is like a separate script in the main script.

Yet another example of a function is this:

```
#!/bin/bash

myvar="$1"

if [ "$myvar" = "" ]; then
    echo "Usage: $0 <number>"
    exit 0
fi

function calcfunc { expr 12 + $1 ; }

myvar2=`calcfunc 5`

echo "welcome"
echo "Now we will calculate 12 + 5 * $myvar"

echo "the answer is `expr $myvar2 '*' $myvar`"
```

Which executed looks like this:
Without any arguments:

```
alien:~$ ./funk.sh
Usage: ./funk.sh <number>
alien:~$
```

And with arguments:

```
alien:~$ ./funk.sh 23
welcome
Now we will calculate 12 + 5 * 23
the answer is 391
alien:~$
```

And for the sake of knowledge it should also be said that a function can be declared in the following ways as well:

```
#!/bin/bash

function foo() {
    echo "hello world"
}

foo
```

```
#!/bin/bash

foo () {
    echo "hello world"
}

foo
```

Note that the parentheses after the function name are the new thing here.
It's used exactly the same way as without the parentheses, I just added that here so that you won't get confused if you see it made that way sometime.

So if you make a function, to call for it (to make use of it), just use the the functions name just as if it had been a command.

If there is anything that is uncertain at this point, go back and read it again, until you understand it, or at least get the basic idea. =)

```
================================================================================
```

4 - Other techniques

=== Table Of Contents

Now let's move on to a little bit more advanced shell scripting.

Actually it's not that advanced, just more hard to keep in order, but let us leave that to the head of the beholder..... errr
Anyway, let's not make this harder than it is, so here we go, with a script example:

```
#!/bin/bash

> file1.c

cat >> file1.c << EOF
#include <stdio.h>
int main ( void )
{
    printf("Hello world\n");
    return 0;
}
EOF

cc file1.c -o file1 || gcc -o file1 file1.c
./file1

rm -f file1.c file1

exit 0
```

And here as follows, is an semi-English translation of the script:

```
echo nothing to file1.c to create it.

cat to file1.c what comes here after in between the "EOF"'s
// --- a short hello world program in C code --- //

try if there is a 'cc', if not then use 'gcc'
Execute the newly compiled file

remove file1.c and file1

exit the script.
```

This can be very useful, since bash does have its limitations, so if you ever need something more powerful or just something else, you can always do like the script told.

Another little trick with the same thing in a script is:

```
more << EOF
Here you can type whatever,
like an agreement text or something.
EOF
```

Play around with it.

Here let's have a look at the `case` command. The `case` command, like `if`, ends with the command backwards.
So that what starts with `case` ends with `esac`. Here's an example of `case`:

```
#!/bin/bash

case "$1" in
    foo)
        echo "foo was written"
        ;;
    bar)
        echo "bar was written"
        ;;
    something-else)
        echo "something-else was written"
        ;;
esac
```

This is the same as saying:

```
....
if [ "$1" = "foo" ];then
    echo "foo written"
fi
if [ "$1" = "bar" ];then
    echo "bar was written"
fi
```

```
etc.
....
```

so `case` is far shorter if you have a lot of arguments.

Here's a better example:

```
#!/bin/bash

case "$1" in
    --help)
        echo "Usage: $0 [--help] [--port <port>] <host> [--time]"
        ;;
    --port)
        telnet $3 $2
        ;;
    --time)
        date
        ;;
esac
```

This is not very hard to learn,

```
case the first argument vector ($1) in
first-possible-match)
if it matches do ........
close the ) with ;;
etc. down to "esac"
```

Really not much more to say about the case command at this point.

Now let's have a REALLY quick look at the command `sed`, ('string editor') which is used to edit and reformat text. Say now that you have a file called `tmp` that contains the following:

```
http://www.metacrawler.com
http://www.yahoo.com
http://www.webcrawler.com
```

and you want to change all the "`www`"'s to "`ftp`", then you do like this:

```
sed 's/www/ftp/g' tmp
```

and if you want to store the changes to a file you can do:

```
sed 's/www/ftp/g' tmp > tmp2
```

This is not `sed`'s only use, but for sure it's what it's most used for.

Here's just one other really simple thing `sed` could be used as:

```
sed -n 3 p /etc/passwd
```

This will print out the 3rd line of the `/etc/passwd` file.

Now let's take up a really interesting command `dialog`, that is a command with which you can create 'ncurses' `dialog` boxes.
Ncurses `dialog` boxes are what one would call 'console graphics' or 'ascii color graphics', if you ever seen a blue background and a gray box asking questions, with an <OK> and <Cancel> button, while running something in a console you have seen an ncurses `dialog` box.

Now here is a small script example of a `dialog` box:

```
#!/bin/bash

dialog --backtitle "My first dialog" \
       --title "Main menu" \
       --menu "Make your choice" 13 60 6 \
        1 "First option" \
        2 "Second option" \
        3 "Exit" 2> .tempfile
        output=`cat .tempfile`
        rm -f .tempfile
```

```
if [ "$output" = "1" ]; then
    dialog --msgbox "First option was entered" 5 40
fi

if [ "$output" = "2" ]; then
    dialog --msgbox "Second option was entered" 5 40
fi

exit 0
```

Here is another very small example with `dialog` boxes:

```
#!/bin/bash

dialog --yesno "Do you agree, jada jada" 10 50 && \
dialog --yesno "ok you agreed" 10 50 || \
dialog --yesno "ok fine, leave then ..." 10 50
---------------------------------------------------------------------------------
In English:
If the first one (Do you agree, jada jada) returns 'true' (yes)
then it walks on to the next (ok you agreed),
and if any of those first two returns 'false' (no) it will display
the last (ok fine, leave then ...).
```

Note:

The back slashes"`\`" are used to say "no new line" as in what comes on the next line will be treated as if it were on the same line as the last line, the "`\`" really means that the next character's special meaning (in this case the new lines) is overlooked.
Just in case you didn't understand, the numbers after, like 10 50:
`dialog --yesno "ok fine, leave then ..." 10 50`
is the geometry of the window. First number is height and the second width.
Another note being that the command "`Xdialog`" works the same as `dialog`, but I won't take that up here because it doesn't come as default with any other Linux distribution than Mandrake, as far as I know.
A final note is that the `dialog` command is getting to be out-dated but is still the most used, the newer version of it is named `whiptail` and works the same as `dialog`, but looks slightly different.

Now we have covered most of it, so let's take up some small tricks, that bash allows you to do, here follows what it does, and then the code example:

Here we wanna check if you have a directory called `/usr/local/bin`:

```
if [ -d /usr/local/bin ]; then
    cp file /usr/local/bin/
else
    echo "NO !!"
fi
```

Another way of doing the same thing is this:

```
test -d /usr/local/bin && cp file /usr/local/bin/ || echo "NO !!"
```

Or:

```
ls /usr/local/bin/ && cp file /usr/local/bin/ || echo "NO !!"
```

The last way is a bit messy, but a lot smaller than the first one, but here's yet another way that is small and less messy:

```
ls /usr/local/bin/ 1>/dev/null 2>&1 && cp file /usr/local/bin/ || echo "NO !!"
```

That might look really weird at first sight, but it's easy if you break it down and look at it:

```
ls /usr/local/bin/ <<==== lists /usr/local/bin/
1>/dev/null        <<==== sends the contents of the listing to 'the black hole'
2>&1               <<==== sends any errors the same way... to 'the black hole'
                          (same thing as to say 2>/dev/null)
&&                 <<==== if the first command worked, we will go on here.
cp file /usr/local/bin/   <<==== copy file to /usr/local/bin/
||                 <<==== if the  first command didn't work...
                          we go on here instead.
echo "NO !!"       <<==== what it says ... say NO !!
```

as this:

```
If `ls` can list /usr/local/bin/ next command can be executed, OR if not
it will echo "NO !!", and all listings/errors are being sent to /dev/null
the 'black hole' of a UNIX/Linux.
```

To prevent that a script is being executed more the once at the same time for some reason you may wanna let the script make a 'lock' file. This is very easy to do:

```
#!/bin/bash
ls script.lock 1>/dev/null 2>&1 && exit 0 && echo "lockfile detected"
> script.lock
echo "Here is where the script should be"
rm -f script.lock
exit 0
--------------------------------------------------------------------------------
In English:
Here we first check if there is a lockfile, and if there is we terminate the
script and say that a lockfile was detected.
If there is no lockfile, we create one and start to execute the rest of the
script.
At the end of the script we remove the lockfile, so that the script can be
executed again.
All this is just to prevent the same script to be run twice at the same time,
which can be a good thing if your script does something that can't be done
twice at the same time, as mounting a hard drive/cd-rom, using sound or
anything like that.
```

Another neat little trick is if you from within a script are going to create temporary files that you want unique (to not overwrite some other files anywhere, wherever the script may get executed), then you can use a predefined variable called `$$`, which is the 'process ID' or 'pid' of the executing shell, like this:

```
#!/bin/bash
echo "ls" >.tmp.$$
echo "-la" >.tmp2.$$
one=`cat .tmp.$$`
two=`cat .tmp2.$$`
$one $two
rm -f .tmp.$$ .tmp2.$$
```

This will make a file called `.tmp.<pid of script>` containing the word "`ls`", then it will make a file called `.tmp2.<pid of script>` containing "`-la`".
After that it makes 2 variables, each one when being called will concatenate ('`cat`') one of the `.tmp.*` files each. At the end we have "`$one $two`" that will work the same as if we had printed:

```
ls -la
```

And last we remove the temporary files.

This is useful if you're doing a script that requires you to move around a lot of text from one file to another and back, as this example:

```
#!/bin/bash

sed 's/www/ftp/g' tmp > .tmp.$$
sed 's/com/org/g' .tmp.$$ > .tmp2.$$
sed 's/ /_/g' .tmp2.$$ > .tmp3.$$
mv -f .tmp3.$$ tmp
rm -f .tmp.$$ .tmp2.$$ .tmp3.$$

exit 0
```

Here we change all `www`'s in a file (`tmp`) to `ftp`, then we change all `com`'s to `org`, and then all spaces to underscores.
After that we move the fully changed file so it overwrites the original file.
Then removing the temporary files and exit the script.

If you have a good look at it, it's really easy.

Another nice trick is as I showed in the example prior to the last one:

```
...
one=`cat .tmp.$$`
two=`cat .tmp2.$$`
...
```

That a variable can hold a command can prove to be useful, like this:

```
#!/bin/bash
time=`date +%H:%M:%S`

echo "$time" >> log
echo "some input to the log" >> log
sleep 60
echo "$time" >> log
echo "some input to the log a minute later" >> log

exit 0
```

But, it can hold more than just a command, it can actually *hold* the contents of a whole file.
Say now that you made a script and have a pretty large readme file, and want to display that as a 'man page' to the script if the argument `--help` is used to execute the script, then you can do like this:

```
#!/bin/bash
one="$1"
help=`cat README`

if [ "$one" = "--help" ]; then
    $help | more
...
```

Of course it would be easier to say:

```
#!/bin/bash
if [ "$?" = "--help" ]; then
    more README
fi
```

But these examples are just here for illustration so you get the point of usage for commands and so.

Another trick is, if you wanna hide script/program you can rename it to: `-bash`, that way it will look as a normal bash running in the `ps` (process list): you rename it by doing:

```
mv script ./-bash
```

Then execute it like normal `./-bash`

Yet another trick, is if you're doing a script where you want each line of a file as a variable, unlike `for` which takes each word as a variable. This can be done like this:

```
#!/bin/bash
file="$1"
min="0"
max=`cat $file | wc -l`

if [ "$1" = "" ]; then
    echo "Usage: $0 <file>"
    exit -1
fi

while [ "$min" != "$max" ]; do min=`expr $min + 1`
    curline=`head -$min $file | tail -1`
    echo $curline
    test $min -eq $max && exit 0
done
```

The `test` is there to make sure that it will end when `$min` and `$max` are the same. Now this can be done with `for` if you change the 'IFS' (described later), but that is not recommended, especially if you export IFS since that would change the environment and hence screw with the system scripts if they were to be run before changing IFS back to normal, but enough about that now, just keep it somewhere far back in your head, don't change IFS unless you know what you're doing.

AL: Although this example is intended to show usage, it actually runs rather slowly, because for each line in the file, there is run `cat`, `wc`, `head` and `tail`. Better is something like `sed -n -e <line number required>p filename`, e.g.:

```
curline=`sed -n -e ${min}p $file`
```

`-n` means suppress printing of pattern space (often used with `p`, explained below)
`-e` means execute the following script
`${min}p` is a script which means '<for this line number>, print'

This still runs through the file once for every line, but only uses `sed`.

If you don't understand this little script at this point, don't worry, you will understand it the second time you read this tutorial =)

Now let's take a quick look at arrays in shell scripting.

First off, an array is what it says, it's an array of something, now, to declare a variable that can hold an array we create it with the command '`declare`', let's make a short example:

```
alien:~$ declare -a foo=(1 2 3 4 5)
alien:~$ echo ${foo[0]}
1
alien:~$ echo ${foo[1]}
2
alien:~$ foo[1]=bar
alien:~$ echo ${foo[1]}
bar
alien:~$
```

First of all, to understand the declare command better do "`help declare`" at a console and it'll display this:

```
declare: declare [-afFrxi] [-p] name[=value] ...
    Declare variables and/or give them attributes.  If no NAMEs are
    given, then display the values of variables instead.  The -p option
    will display the attributes and values of each NAME.

    The flags are:

      -a        to make NAMEs arrays (if supported)
      -f        to select from among function names only
      -F        to display function names without definitions
      -r        to make NAMEs readonly
      -x        to make NAMEs export
      -i        to make NAMEs have the `integer' attribute set

    Variables with the integer attribute have arithmetic evaluation (see
    `let') done when the variable is assigned to.

    When displaying values of variables, -f displays a function's name
    and definition.  The -F option restricts the display to function
    name only.

    Using `+' instead of `-' turns off the given attribute instead.  When
    used in a function, makes NAMEs local, as with the `local' command.
```

So here we see that the `-a` switch to declare makes the variable an array.
So after getting that '`declare -a`' we declare the variable as an array, with the array within parentheses.

And then to make use of it, we use the way to write a variable like this:
`${variable name here[number]}`
and the number inside the `[]`'s is the number that points to which part of the array it should use, beginning from `0` which is the first.

Let's make another short example:

```
declare -a foo=(this is another example)
echo "The array (${foo[*]}) has (${foo[0]}) as first, and (${foo[3]}) as last."
```

The output of this would be:

```
The array (this is another example) has (this) as first, and (example) as last.
```

Now, this isn't something you'll use in every day scripting, but it's still something you should know the existence of, just in case you see it or need it at some point.

Now here's a less common way of using bash: CGI scripts.
Most people don't associate shell scripting with cgi, but it works just as well as any other language, so here I'd like to show you how to make CGI scripts in bash.

Here is the first example which is a simple cgi counter in bash.
A note is that all CGI scripts should be in the servers cgi-bin directory or any subdirectory there off, unless the server is configured to see any

other directories as cgi directories.

```
#!/bin/bash

test -f date.txt || echo `date "+%B %d %Y"` > date.txt
test -f counter.txt || echo '0' > counter.txt
current=`cat counter.txt`
date=`cat date.txt`
visitor=`expr $current + 1`

echo "$visitor" > counter.txt
echo 'Content-type: text/html'
echo ''
echo '<br>Vitor:'
echo '<br>'$visitor'<br>Since'
echo '<br>'$date'</br>'
```

Let's take this one line by line here:
First the shell ....

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

Then we test if there is a file called `date.txt`, if not then we echo the current date to it and hence creating it.

```
test -f date.txt || echo `date "+%B %d %Y"` > date.txt
```

Then we test if there is a file called `counter.txt` and if not we echo a `0` to it and so create that one too.

```
test -f counter.txt || echo '0' > counter.txt
```

Now we declare the variables, current is the contents of `counter.txt`.

```
current=`cat counter.txt`
```

The `date` variable is the contents of `date.txt`.

```
date=`cat date.txt`
```

And `visitor` is the sum of the contents of `counter.txt` + 1.

```
visitor=`expr $current + 1`
```

And then we echo the new increased number to `counter.txt`.

```
echo "$visitor" > counter.txt
```

And here comes the HTML part. the first top line is the 'cgi header': that should ALWAYS be there:

```
echo 'Content-type: text/html'
echo ''
```

Then we move on to the *real* html:

```
echo '<br>Vitor:'
echo '<br>'$visitor'<br>Since'
echo '<br>'$date'</br>'
```

The `<br>` is a linebreak in html
The bash variables have to be *outside* the 's else they will simply show up as `$visitor` or `$date` literally, that is why it's made like this:

```
echo 'text' $variable 'some more text'
```

So that the text is enclosed with 's, but the variables are between or rather outside of them.
Anyway, this cgi will create a section that looks like this on a webpage:

```
   Vitor:
     1
   Since
May 29 2001
```

To add that to a html code you add this tag to your html/shtml page:

```
<!--#exec cgi="<path to counter>" -->
```

With the path to the counter it could look like this:

```
<!--#exec cgi="/cgi-bin/counter/counter.cgi" -->
```

Not so hard is it?

Here is another example of a CGI script in bash (actually the second CGI script I ever made).

```
#!/bin/bash

method=`echo $QUERY_STRING | awk -F'=' '{print $1}'`
host=`echo $QUERY_STRING | awk -F'=' '{print $2}'`

if [ "$method" = "nslookup" ]; then
    echo 'Content-type: text/html'
    echo ''
    echo '<html>'
    echo '<body bgcolor="white">'
    echo '<center>'
    echo '<br>nslookup '$host' (This might take a second)<br>'
    echo '<hr width="100%">'
    echo '</center>'
    echo '<pre>'
    nslookup $host
    echo '</pre>'
    echo '<center>'
    echo '<hr width="100%">'
    echo '<br>nslookup compleat'
    echo '</center>'
    echo '</body>'
    echo '</html>'
fi

if [ "$method" = "ping" ]; then
    echo 'Content-type: text/html'
    echo ''
    echo '<html>'
    echo '<body bgcolor="white">'
    echo '<center>'
    echo '<br>ping '$host' (This might take a second)<br>'
    echo '<hr width="100%">'
    echo '</center>'
    echo '<pre>'
    ping -c 5 $host
    echo '</pre>'
    echo '<center>'
    echo '<hr width="100%">'
    echo '<br>ping compleat'
    echo '</center>'
    echo '</body>'
    echo '</html>'
fi

if [ "$method" = "scan" ]; then
    echo 'Content-type: text/html'
    echo ''
    echo '<html>'
    echo '<body bgcolor="white">'
    echo '<center>'
    echo '<br>Scanning host '$host' (This might take a minute)<br>'
    echo '<hr width="100%">'
    echo '</center>'
    echo '<pre>'
    nmap $host
    echo '</pre>'
    echo '<center>'
    echo '<hr width="100%">'
    echo '<br>Scan compleat'
    echo '</center>'
    echo '</body>'
    echo '</html>'
fi
```

Now let's take a look at what that means:

This time it won't be all the lines, but all the new parts:

First the 2 variables:

```
method=`echo $QUERY_STRING | awk -F'=' '{print $1}'`
host=`echo $QUERY_STRING | awk -F'=' '{print $2}'`
```

These are made this way because of how the CGI script imports the variables from a form (I'll come back to this), the `$QUERY_STRING` variable is from the webservers environment, and so is one of the `httpds` env variables.

And what you do with the `$QUERY_STRING` is depending on how you create your web form .... but as I said I'll get back to that.

Now the rest:

```
if [ "$method" = "nslookup" ]; then
```

That was pretty obvious .... if the first field of `$QUERY_STRING` (separated by a `=`, is `nslookup`, then go ahead here:

```
echo 'Content-type: text/html'
echo ''
```

Yes the header ....

```
echo '<html>'
echo '<body bgcolor="white">'
echo '<center>'
echo '<br>nslookup '$host' (This might take a second)<br>'
echo '<hr width="100%">'
echo '</center>'
echo '<pre>'
```

Create a HTML page ... and then after the `<pre>` we do the actual center part of the script:

```
nslookup $host
```

Which will resolve the DNS of the host (try the command and see), And after that we end the html page:

```
echo '</pre>'
echo '<center>'
echo '<hr width="100%">'
echo '<br>nslookup compleat'
echo '</center>'
echo '</body>'
echo '</html>'
```

and then end the `if` statement:

```
fi
```

and then the same for the others, just different objects at what they should do, as this was nslookup, the other sections will mnap (portscan) and ping the host instead.

Now how would a full HTML page look to make use of this cgi script?
As we this time need input to get the host or IP to scan/ping/nmap.

Well like this:

```
<html>
    <body bgcolor="white">
        <center>
        <p><font size="+1">Enter host or IP</font></p>
        <hr width="100%">
        <br>
        <form action="http://www.yourdomain.com/cgi-bin/scan.cgi" method="get">
            <input type="text" name="scan" value="" size="30">
            <input type="submit" value="portscan">
        </form>
        <p>
        <form action="http://www.yourdomain.com/cgi-bin/scan.cgi" method="get">
```

```
            <input type="text" name="nslookup" value="" size="30">
            <input type="submit" value="nslookup">
        </form>
        <p>
        <form action="http://www.yourdomain.com/cgi-bin/scan.cgi" method="get">
            <input type="text" name="ping" value="" size="30">
            <input type="submit" value="ping -c 5">
        </form>
        </center>
    </body>
</html>
```

Now what does all this mean? ....

Well, I won't turn this into a HTML tutorial, but I'll explain this so you can make use of bash for CGI.

Right to the important HTML part here:

```
        <form action="http://www.yourdomain.com/cgi-bin/scan.cgi" method="get">
            <input type="text" name="scan" value="" size="30">
            <input type="submit" value="portscan">
        </form>
```

Here we create a form, as in an input field, which will add its input (in a specific way) to the end of the url in action="".
The method is `get` since we're getting the output of the cgi script.
We name this field scan so we get the output this way:

```
scan=<input>
```

Where the `<input>` is what you typed in the input box.
And then we make an "ok" button that says "portscan".
So if you type say `127.0.0.1` and press the portscan button the URL it will be directed to is this:

```
http://www.yourdomain.com/cgi-bin/scan.cgi?scan=127.0.0.1
```

And this "`scan=127.0.0.1`" will be the `$QUERY_STRING` environmental variable.

And so the script is starting to make sense.

Here's a REALLY simple cgi script just for the illustration as well.

```
#!/bin/bash

string="Hello World"

echo 'Content-type: text/html'
echo ''
echo '<html>'
echo '<br>'$string'</br>'
echo '</html>'
```

And the html to call that ..... just a normal hyper link.

```
<a href="http://www.yourdomain.com/cgi-bin/yourscript.cgi">Link</a>
```

And that is it.

That is it on the tricks, now let's move on to practical examples so you get a little bit of feel for how you can use bash to make things easier for you.

===

5 - Practical Scripting Examples

== Table Of Contents

I'd first like to add a note which you already probably know: "`./`" means look in current directory instead of the "`PATH`".

To give that an example, say now that you have a script in your home directory called `ls` or `dir`, how would you execute that without getting the contents of the directory? Well, that is why you use "`./`" before a name to execute it if it's in the current directory.

"../" is the previous directory (one directory further up towards "/" than you are currently in), this can be used as this, say that you have a script called "script" in "/home/user/" and you are standing in "/home/user/nice/" and you don't want to leave the directory but still want to execute the script.

Then you do, "../script" and if you are in "/home/user/nice/blah/" you would do, "../../script". "../../" means 2 directories back. Get the idea?

Anyway, now to the practical examples, which are working scripts for various purposes, to give an idea about what you can do with shell scripting. New things previously not explained will show up in this section, but I will explain them as we go along.

Let's start with simple examples and move on to harder parts. As for first I'll stick to useless scripts =) just for illustration. Explanation on the scripts follow after them, as usual. So here we go on that.

```
#!/bin/bash

one="$1"
something="$2"

if [ "$one" = "" ]; then
    echo "Usage: $0 [name] <anything goes here>"
    exit 0
fi

function first {
    clear
    echo "this is a test script !"
    echo
    echo "name followed on $0 was - $one - "
    echo
    echo "if you typed anything after the name it was: $something"
    echo
}

first

exit 0
```

Executed without any thing after the script name it looks like this:

```
alien:~$ ./script
Usage: ./script [name] <anything goes here>
alien:~$
```

Executed with a name it looks like this:

```
alien:~$ ./script Jerry
----------<on a cleared screen>----------
this is a test script !

name followed on ./script was - Jerry -

if you typed anything after the name it was:

alien:~$
```

Executed with a name and something else it looks like this:

```
alien:~$ ./script Jerry homer
----------<on a cleared screen>----------
this is a test script !

name followed on ./script was - Jerry -

if you typed anything after the name it was: homer

alien:~$
```

Notes:

`$0` is the script name's variable so you can do a "`Usage: <scriptname>`" regardless of whether the script is renamed after you made it.
`$1` is the first thing that is typed after the script in the command line.
`$2` is the second thing that is typed after the script in the command line.
`$3`, `$4` and soon .....
`one="$1"` this puts the contents of "`$1`" into the variable `$one` which can be very useful to avoid errors.
`clear` clears the screen.

This next example is a script which you really shouldn't use... It's here as an example for a working help script, but *could* cause harm if not used correctly. It runs through each user's `.bash_history` file looking for the string "`passwd`", and then offers you the choice of deleting the user. So if you lose anything because of using it, it's all on you. and don't say I didn't warn you.

```
#!/bin/bash

if whoami | grep -v root >> /dev/null; then
    echo "you have to be root to use this"
    exit 1
else

    cat /etc/passwd | cut -f1 -d : | grep -v halt | grep -v operator | \
        grep -v root | grep -v shutdown | grep -v sync | grep -v bin | \
        grep -v ftp | grep -v daemon | grep -v adm | grep -v lp | \
        grep -v mail | grep -v postmaster | grep -v news  | grep -v uucp | \
        grep -v man | grep -v games | grep -v guest | grep -v nobody > user.list
fi

for USER in `cat user.list`; do
    if cat /home/$USER/.bash_history | grep passwd >> /dev/null; then
        echo
        echo "user $USER have tried to access the passwd file"
        echo "do you want to remove $USER from your system [y/n] "
        read YN
        if [ "$YN" = "y" ]; then
            echo "user $USER is being deleted"
            echo "home dir of user $USER is however intact"
            echo
            remuser $USER
        else
            echo "user $USER is not deleted"
            echo
        fi
    else
        echo "$USER haven't tried to access the passwd file"
    fi
done

rm user.list
echo
echo "Script finished"
echo
exit 0
```

I will just translate this script into real/clear English:

```
if (check own user-name) is anything else but root >> send output to a black hole
say, "you have to be root to use this"
terminate program.
in other cases (in this case that can only be if the user is root)

list the contents of the file "/etc/passwd" combined with - cut out the user names (field 1 separated by ":")
grep everything except lines containing the following words/names:
    halt operator root shutdown sync bin ftp daemon adm lp mail postmaster news uucp man games guest nobody
and send it to the file "user.list"
end "if" statement

for each user in the "user.list" file do the following
if the word "passwd" is present in "/home/$USER/.bash_history" >> output to
the system's black hole
say nothing
say "user $USER has tried to access the passwd file"
say "do you want to remove $USER from your system [y/n]"
read if the input from the keyboard is a "y" or "n"

if the variable for the answer of the input is "y" then
say "user $USER is being deleted"
say "home dir of user $USER is however intact"
say nothing
removing the user from the system that tried to access the passwd file
in other cases
say "user $USER is not deleted"
say nothing
end "if" statement
in other cases
say $USER haven't tried to access the passwd file
end "if" statement
exit the for-loop

remove the "user.list" file
```

```
say nothing
say "Script finished"
say nothing

exit and return to the shell command line.
```

Note: `grep -v` means show every line which does *not* contain the string after the `-v`.

Here is another way of doing the exact same script, just to illustrate that the same thing can be done in several different ways. This script also offers you the choice of deleting a user, so again, if you lose anything because of using it, it's all on you. and don't say I didn't warn you.

```
#!/bin/bash

if [ "$UID" != "0" ]; then
    echo "you have to be root to use this"
    exit -1
fi

for uids in `cat /etc/passwd`; do
    useruid=`echo "$uids" | awk -F':' '{print $(3)}'`
    test "$useruid" -ge "500" 2>/dev/null &&
    echo "$uids" | awk -F':' '{print $(1)}' > user.list
done

for USER in `cat user.list`; do
    if (grep passwd /home/$USER/.bash_history >/dev/null); then
       echo
       echo "user $USER have tried to access the passwd file"
       echo "do you want to remove $USER from your system [y/n] "
       read YN

       case $YN in
            y* | Y*)
               echo "user $USER is being deleted"
               echo "home dir of user $USER is however intact"
               remuser $USER
               echo
               ;;
            n* | N*)
               echo "user $USER is not deleted"
               echo
               ;;
       esac
    else
       echo "$USER haven't tried to access the passwd file"
       rm -f user.list
       echo
       echo "Script finished"
       echo
    fi
done

exit 0
```

Since this script does the exact same thing, but in another way, I'll leave you with the experience of trying to figure out the differences and how it works with the help of this tutorial, you might not get this right until you've read this tutorial twice.

A tip is: try to make a comment to each line with what it does and why.

This below script is a "Wingate" scanner, to scan for wingates that can be used to bounce off and such things, don't know if that is really legal so take that on your own risk.

Anyway here comes the script:

```
#!/bin/bash
echo > .log.tmp
echo > .log2.tmp
echo "sleep 7" > wg.config
echo "killall -2 telnet" >> wg.config

scan="$1"
count="0"
max="255"

clear
```

```
if whoami | grep root >> /dev/null ; then
   echo "please use this as user and not root, since it would kill all users"
   echo "telnet sessions"
else
   clear
fi

if [ "$1" = "" ]; then
    echo " usage is: $0 <C host> "
    echo "  examples:"
    echo "  $0 127.0.0"
    echo " That will scan from 127.0.0.0 to 127.0.0.255"
    echo
    echo "be aware though, while it scans it also kills any other telnet"
    echo "sessions you might have ...."
    exit 0
fi

while [ "$count" != "$max" ]; do count=`expr $count + 1`
    echo "Attempting connection to $1.$count "
    echo > .log2.tmp
    ./wg.config &
    telnet $scan.$count >> .log.tmp
    cat .log.tmp | grep -v refused | grep -v closed | grep -v Connected | grep -v Escape | grep -v login >> .log2.tmp
    echo >> .log.tmp
done
echo "Script Finished, results stored in .log.tmp and .log2.tmp"
exit 0
```

This time I will not translate the script into clear English and I will not show how it looks executed, I leave that for you to do =)

Now a final practical example of a script, this is a small graphical front end to the console mp3 player `mpg123` so you got to have that to work and you need to execute this script in a directory where you have mp3's ....

Also if you want the X-windows part of it to work you need to get and install Xdialog, you can get that from www.freshmeat.net ...

However if you have Linux Mandrake you should be good anyway, Xdialog comes as default in Mandrake.

This script should be named `xmpg123`. So here we go:

```
#!/bin/bash
dialog --backtitle "xmpg123" \
       --title "Main menu" \
       --menu "Make your choice" 13 60 6 \
        1 "X-windows" \
        2 "Ncurses" \
        3 "Exit" 2> .tempfile
output=`cat .tempfile`
echo $output
rm -f .tempfile

if [ "$output" = "1" ]; then
    for songs in `ls -1 *.mp3`; do
        echo "$songs mp3-file" >> .tempfile
    done
    output=`cat .tempfile`
    Xdialog --menu 2 2 2 1 $output  \
        2> .tempfile
    output=`cat .tempfile`
    mpg123 $output
    rm -f .tempfile
fi

if [ "$output" = "2" ]; then
    for songs in `ls -1 *.mp3`; do
        echo "$songs mp3-file" >> .tempfile
    done
    menu=`cat .tempfile`
    dialog --menu "Make your choice" 13 70 6 $menu 2> .tempfile
    output=`cat .tempfile`
    mpg123 $output
    rm -f .tempfile
fi

exit 0
```

A note being that dialog and Xdialog seems to be in early stages, so this may look sort of buggy if you don't have the great dialog god at your side...

And don't forget to "`chmod u+x <script name>`" or "`chmod a+x <script name>`" to make your scripts executable.

==

6 - Customize your system and shell

== Table Of Contents

This section is dedicated to how you can customize your system in various ways, this section was never planned to be in this tutorial, but since I have received so many questions on how to do all this, I might as well include it in the tutorial.

First of I'm going to explain the local settings, this means the settings that will only affect a single user and not the whole 'global' system. And the most logical way to start this is (I think) to talk about the shell.

At the very top of this tutorial you will find the various types of shells, default for most systems is `/bin/bash`, this is set in the `/etc/passwd` file so a normal user can not change that.

What a normal user can do if he wants to use another shell is to start it from his `~/.bashrc` file.

So say now that you feel the urge to run tcsh, then just add the line `/bin/tcsh` in your `~/.bashrc`, this may be done by doing:

```
echo "/bin/tcsh" >> ~/.bashrc
```

personally I prefer the standard bash.

But if you do have root (super user) access to the system, you can change the shell type correctly in the `/etc/passwd` file.

here's a user account with `/bin/bash` from `/etc/passwd`.

```
User:x:500:500:my real name:/home/user:/bin/bash
```

And here the same line changed to `/bin/tcsh` (tenex C shell)

```
User:x:500:500:my real name:/home/user:/bin/tcsh
```

Here are the system variables you can use to change your environment, these can be set and exported from your `~/.bash_profile` or `/etc/profile`. It's not all of the variables but all the really interesting ones, so here we go:

`BASH=`
: this can also set your shell type, it's most commonly defaulted to `BASH=/bin/bash`

`BASH_VERSION=`
: this can change the version reply of bash, on my system this is defaulted to `BASH_VERSION='2.03.19(1)-release'`

`ENV=`
: this should point to a file containing your environment, this is by default: `ENV=~/.bashrc`

`HISTFILE=`
: this should point to a file that will contain your shell 'history', as in your previously used commands. this is by default set to: `HISTFILE=~/.bash_history`

`HISTFILESIZE=`
: the max allowed size of the history file, usually around 1000

`HISTSIZE=`
: about the same as `HISTFILESIZE`

`HOME=`
: this should point to your home dir

`HOSTNAME=`
: this is your hostname

`HOSTTYPE=`
: this should return the same as the `arch` command.

`IFS=`
: 'Internal Field Separator' .... this is a delimiter, often defaulted to a new line as this:

```
......
IFS='
'
......
```

`INPUTRC=`
: defaulted to `INPUTRC=/etc/inputrc`

`LANG=`
: language variable, default is `en` for English

LANGUAGE=
: about the same as LANG, also defaulted to `en` for English

LINGUAS=
: defaulted to `en_US:en` also a language variable.

LS_COLORS=
: sets colors to the `ls` command. This on my system is defaulted to this:

```
......
LS_COLORS='no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:\
cd=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=01;32:*.cmd=01;32:*.exe=01;32:\
*.com=01;32:*.btm=01;32:*.bat=01;32:*.tar=01;31:*.tgz=01;31:*.tbz2=01;31:\
*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lha=01;31:*.zip=01;31:\
*.z=01;31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:*.bz=01;31:*.tz=01;31:*.rpm=01;31:\
*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.xbm=01;35:*.xpm=01;35:\
*.png=01;35:*.tif=01;35:*.tiff=01;35:'
......
```

So just add what you want here, and colors are the same as explained about how to set the prompt later down, but without the `[` in front and `m` on the end.

MAIL=
: mail file, usually `MAIL=/var/spool/mail/<username>`

OSTYPE=
: This can change your OS reply, it's on a linux defaulted to: `OSTYPE=linux-gnu`

PATH=
: changes your path, this variable is explained in the explanation of the `/etc/profile` file

PPID=
: parent pid .... this is a read-only variable ... so you can't change it.

PS1=
: the prompt variable, explained later down.

PS2=
: the *more to come* variable, as if you type an unfinished command string, it will bring you a new prompt where you can finish it, this is usually defaulted to: `PS2='> '`

SHELL=
: another way to change your shell type ...

TERM=
: terminal type, usually defaulted to: `TERM=linux` but can also be like: `TERM=vt100` there are more video terminals than 100 though.

UID=
: your user ID, if you're root this will be `0`, this is a readonly variable.

USER=
: your user name ....

USERNAME=
: same as `$USER`

Say that you want to change your shell to `/bin/csh` and your path to just `/bin` (you don't), but just if you would in your: `.bash_profile` add:

```
SHELL=/bin/csh
PATH=/bin

export PATH SHELL
```

Not so hard huh?

The next thing here is a question that I've heard a lot, and that is "how do I change my command prompt?".

The command Prompts variable is named `PS1` (`$PS1`). For a prompt that looks like this:

```
[alien@localhost alien]$
```

the contents of the `PS1` variable would be this:

```
[\u@\h \W]\$
```

All the prompt's internal variables start with a `\` (backslash).

Useful:

```
\$ = the prompt ($ for user and # for root)
\d = date
\h = abbreviated hostname (root.host.com would become root)
\H = full hostname
\s = shell type
\t = time
\T = time with seconds
\u = username
\v = shell version (short)
```

```
\V = shell version (long)
\w = full path
\W = current directory name
```

less useful:

```
\e = erase rest of line .... not very useful
\n = new line ... not very useful
\r = whatever comes after \r is displayed before the first character.
```

A couple of examples would be:

*BSD like.

```
PS1="\u: \w> "
```

DOS like.

```
PS1="C:\w > "
```

RedHat like.

```
PS1="[\u@\h \W]\$ "
```

Init 1 like.

```
PS1="\s-\v \$ "
```

How do I use colors in the prompt?

To use colors in the prompt you need to be familiar with the escape sequence color codings, those are as follows:

```
reset          = ^[[0m
flashing       = ^[[5m
black          = ^[[0;30m
red            = ^[[0;31m
green          = ^[[0;32m
yellow         = ^[[0;33m
blue           = ^[[0;34m
magenta        = ^[[0;35m
cyan           = ^[[0;36m
white          = ^[[0;37m
highblack      = ^[[1;30m
highred        = ^[[1;31m
highgreen      = ^[[1;32m
highyellow     = ^[[1;33m
highblue       = ^[[1;34m
highmagenta    = ^[[1;35m
highcyan       = ^[[1;36m
highwhite      = ^[[1;37m
bg-white       = ^[[47m
bg-magenta     = ^[[45m
bg-blue        = ^[[44m
bg-red         = ^[[41m
bg-black       = ^[[40m
bg-green       = ^[[42m
bg-yellow      = ^[[43m
bg-cyan        = ^[[46m
```

Important to know is that the leading "`^[`" is NOT 2 characters, it's ONE control character that shows up as 2 characters when you edit the file, or is described by 2 characters. When editing, if you have a real `^[` and you try to delete the `[` it will delete both the `[` and the `^` at the same time.

Not really sure where to put this note but here,

```
^[[<number>G
```

Puts the cursor at column <number>, as this:

```
echo -n "Starting myprog:" && echo -e "^[[50G OK" || echo -e "^[[50G FAILD"
```

So how do you get a real control character?

Either you use the very old line editor `ed` and press `Ctrl+[` to get the control character (`ed` commands are described at the end of this tutorial), or you can use `emacs` or the text editor `joe`.

To get control characters in `emacs`you press `^Q` and `^<what you want>`, as if you want a `^[` you press `^Q^3`, and then `^X^S^X^C` to save and quit.

To get control characters in `joe` you press ` and then the character to make a control character, in this case `[`; when you do this in `joe` the `^[` should like like a bold `[`.

To save and quit in `joe` you press: `Ctrl+K` followed by `Ctrl+X`

It's only the `^[` that is a control character: the rest is normal `[`'s and numbers and so on.

Don't forget to enclose all your colors codes in `\[ \]`; this means that `^[[0;31m` (red) would be written as `\[^[[0;31m\]`.

Where do I write this and how does an example look?

You add this in your `~/.bash_profile`, you can put it at the end of the file.

Some examples are:

```
[ highblue-user red-@ highblue-host green-dir ] highblue-$
PS1="\[^[[1;34m\u^[[0;31m@^[[1;34m\h ^[[0;32m\W^[[0m\]^[[1;34m\$ \[^[[0m\] "

highblue-user highwhite-: highblue-path >
PS1="\[^[[1;34m\]\u\[^[[1;37m\]: \[^[[0;31m\]\w \[^[[0m\]> "
```

(you can not cut and paste these examples without editing the `^[`'s to real control characters, and know that a color prompt is almost always buggy)

The next thing to take up is how to set aliases and to change system variables.

An alias is set in the `~/.bashrc` file if you use `/bin/bash` else, it's most likely in your `.'shell type'rc`, e.g. as `.zshrc`, `.csh`, `.tcsh`, etc.

An alias means that you can make a short command for a longer command, as the alias "`l`" can mean "`ls`" and the alias "`la`" can mean "`ls -la`', and so on. An alias is written like this (also a list of useful aliases):

```
alias rm='rm -i'
alias mv='mv -i'
alias cp='cp -i'
alias s='cd ..'
alias d='ls'
alias p='cd -'
alias ll='ls -laF --color'
alias lsrpm='rpm -qlp'
alias lstgz='tar -tzf'
alias lstar='tar -tf'
alias mktgz='tar -cfz'
alias untar='tar -vxf'
alias untgz='tar -vzxf'
```

These are:
`rm` will now ask before removing anything.
`mv` will now ask before overwriting anything.
`cp` will now ask before overwriting anything.
`s` will now work as `cd ..`
`d` will now work as `ls`
`p` will now work as `cd -` (takes you to your last dir. I.e. you are in `/usr/local/bin/` and move away by doing a `cd /`, if you from here wanna go back to `/usr/local/bin/` you simply type `cd -`, or now just `p`.)
`ll` will do a `ls -la` with colors and a `*` after executable files and a `/` after directories.
`lsrpm` will list the contents (where the files will end up if you install it) of any `.rpm` file.
`lstgz` will list the contents of a `.tar.gz` or `.tgz` file.
`lstar` will list the contents of a plain `.tar` file.
`mkgz` will make a tgz archive (`mktar archive.tgz` directory).
`untar` will untar a `.tar` file.
`untgz` will unzip and untar a `.tar.gz` or a `.tgz` file.

There is more alias like things you can set in the `~/.bashrc` file, like smaller functions that works as aliases, like this:

```
function whichrpm { rpm -qf `'which' $1`; }
```

Typing `whichrpm <command>` at a prompt will name the rpm file it came with.

The `rpm -qf` command works like this:

```
alien:~$ rpm -qf /usr/bin/dir
fileutils-4.0i-1mdk
alien:~$
```

And the function works like this:

```
alien:~$ whichrpm dir
fileutils-4.0i-1mdk
alien:~$
```

`function` - tells bash that its function.
`whichrpm` - user defined name of the function.
`{` and `}` - starts and ends the function.
`rpm -qf` – command
`` ` `` command quote
`'` precise quote
`which` - command to locate a file in your path
`$1` - first argument after the function (the command after the function name when you execute it).
`;` - end line
``function whichrpm { rpm -qf `'which' $1`; }``

So when you execute this, the system will think: Aaah, a function within those `{}`'s, which I should call for when I see the word `whichrpm`, and what's after that word (`$1`) will be used as argument to `which`, and what that returns will be used after `rpm -qf`, which works like this:

```
alien:~$ which dir
/usr/bin/dir
alien:~$
```

So ``` `'which' $1` ``` (when executed with the word `dir`) returns `/usr/bin/dir`, and so the whole function will finally execute: `rpm -qf /usr/bin/dir`

Now more about the files in `/etc`, here you can't be user anymore; to edit the files in `/etc` requires you to be root.

First here I'm going to talk about the `/etc/crontab` configuration file.

The `/etc/crontab` is the global system file for `cron` jobs. `cron` is short for chronological, and as the name tells it is doing things in a chronological order, as you can tell it to run a script or a program once every 1 minute, or you can tell it to run something annually, and anything in between.

On RedHat like systems you have the dirs:

```
/etc/cron.daily/
/etc/cron.hourly/
/etc/cron.monthly/
/etc/cron.weekly/
```

Any script or program that lives in those files will execute by the last name of the file, as if you put a script in /etc/cron.weekly/, the script will execute weekly.

The `/etc/crontab` file looks like this:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

The `SHELL` determines what shell that should be used to execute the things in the `crontab`.

The `PATH` determines what directory's it should look in for commands or programs if no other specific path is given to a program, command or script.

The `MAILTO` variable determines to what user `cron` should send mails to on errors.

And the `HOME` variable determines `cron`'s root/home/base directory.

The first 5 fields determines when to run a job, here's what they mean:

```
Field 1: minutes (0-59)
Field 2: hours (0-23)
Field 3: day of month (1-31)
Field 4: month (1-12 - or names)
Field 5: weekday (0-7 - 0 or 7 is Sun, or use names)
```

The next field is the user that owns the execution process.

Then we have run-parts, and after that the file to execute. (if the file to execute is a dir, it will execute everything in it)

To use the crontab as a user (root included) simply type: `crontab -e` This brings you to a VI like editor (see VI commands later in this tutorial). Say now that you have a script `/home/user/check.sh` that you wanna run every 5'th minute. then you type `crontab -e`. Press the 'Esc' key, followed by `o` to get to "insert" or "edit" mode. In there make the following line:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * *       /home/user/check.sh --flag
```

Then press 'Esc' followed by `:` and then type `wq` followed by enter to write/save and, quit the file, and that is it.

When you run crontab as user you don't have to specify what user that should own the process, `"* * * * * file"` should be enough.

Another way of writing:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * *       /home/user/check.sh --flag
```

Is this:

```
0-59/5 * * * *       /home/user/check.sh --flag
```

That means do this (`/home/user/check.sh --flag`) from 0-59 with 5 as an interval. This means that:

```
* 0-23/2 * * *       /home/user/check.sh --flag
```

Would run the same script every other hour.

Not very hard is it?

Then we have the `/etc/fstab` file which is a list of the HD partitions the system should mount as what when the system boots. This may look like this:

```
/dev/hda1               /                       ext2    defaults        1 1
/dev/hda3               none                    swap    sw              0 0
/dev/hda4               /home                   ext2    defaults        1 2
/dev/hda6               /tmp                    ext2    defaults        1 2
/dev/hdc1               /windows                vfat    defaults        0 0
/dev/fd0                /mnt/floppy             auto    noauto,nosuid   0 0
/dev/hdb                /mnt/cdrom              auto    noauto,ro       0 0
```

First it's the HD partition, then (a few tabs in) the mount point (where the eventual contents of the HD partition should end up), then what file system the partition has, further is if it should be mounted by default etc. and the last 2 numbers is fs_freq and fs_passno (see the man page for fstab).

The possible HD partitions you have to find for your self or know... a tip is to go over the HD's with fdisk, and check for free space.

The possible mount points is only limited by your imagination, though there must always be a `/`. A good disk usage should have these partitions:

```
/       5%
/usr    30%
/home   30%
/tmp    10%
```

And 25% over for other partitions, like `/sources`, or whatever.

The possible and supported file systems are currently:
`minix`, `ext`, `ext2`, `xiafs`, `msdos`, `hpfs`, `iso9660`, `nfs`, `swap`, `vfat`, and perhaps `ntfs`.

The possible mount options are:
`sync`, `user`, `noauto`, `nosuid`, `nodev`, `unhide`, `user`, `noauto`, `nosuid`, `exec`, `nodev`, `ro` etc.

see the man mount page.

So say that you are going to add another cdrom that you want user to be able to mount, and the cdrom is on `/dev/hdd`, then the line to add would look like this (make sure you have the mount point dir, like here you have to mkdir /cdrom):

```
/dev/hdd                /cdrom                  auto    noauto,user,ro  0 0
```

And that is about it for the /etc/fstab

Now I'd like to explain one of the very important files, the `/etc/profile` file. In this file is the Global profile settings, that will apply for all users.

First in this file we should have the `PATH` variable. The directories in the `PATH` are the directories the system will look in if you type a command, for that command to execute it.

Say now that your path looks like this:

```
PATH="$PATH:/usr/X11R6/bin:/bin"
```

And you type `ls`, then the system will first look in `/usr/X11R6/bin` if it can find any file named `ls`, and if it doesn't find it there, it will look in `/bin`, and if it finds it there it will execute it.

The most common places on a system to store commands and programs is in these directories:

```
/usr/X11R6/bin
/bin
/sbin
/usr/bin
/usr/sbin
/usr/local/bin
/usr/local/sbin
```

A path with all those directories in it would look like this:

```
PATH="$PATH:/usr/X11R6/bin:/bin:/sbin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin"
```

The next thing in there can/should be the `PS1` (the prompt), I've already taken up how to customize the prompt, so no need to do that again.

Then (at least in RedHat like systems) we have this:

```
[ "$UID" = "0" ] && {
ulimit -c 1000000
                } || {
ulimit -c 0
}
```

This says: if the UID of the user is `0` (root) then do: `ulimit -c 1000000` or if that doesn't work, do: `ulimit -c 0`.

Then we have an if statement about umask on the user...

After that we define some system variables, where after we export them.

Then we load all the `.sh` scripts in `/etc/profile.d/`

And that is it, in that file.

This is an important file if you wanna add any system variables, or if you want to change anything globally for the whole system.

Now on to the `/etc/hosts.allow` and `/etc/hosts.deny` files.

Those hosts who are in `hosts.allow` are always allowed to connect to the system under the condition that they have valid login and password of course.

Those hosts who are in `hosts.deny` can never establish a lasting connection to your system, even if they have a valid login and password.

If you don't want anyone to connect to your computer at all, you simply add the following to `/etc/hosts.deny`:

```
ALL: ALL
```

And this to `/etc/hosts.allow`:

```
ALL: LOCAL
```

Or if you have a network, you may wanna add this in `/etc/hosts.allow`:

```
ALL: LOCAL, 192.168.1.
```

Where `192.168.1.` is your network call C network.

`/etc/hosts.allow` and `/etc/hosts.deny` understands the following wildcards:

`ALL` The universal wildcard, always matches.
`LOCAL` Matches any host whose name does not contain a dot character.
`UNKNOWN` Matches any user whose name is unknown.
`KNOWN` Matches any user whose name is known.
`PARANOID` Matches any host whose name does not match its address.

Read `man hosts.allow` or `man hosts.deny` (should be the same man file), to find out more about this.

Next up is the `/etc/inputrc` file, which contains brief keyboard configurations. If you want to something like `Ctrl+W` or something to a function of any kind here is the place to do that. The example from the file looks like this:

```
# Just a little example, how do you can configure your keyboard
# If you type Control-x s it's add a su -c " " around the command
# See the info files (readline) for more details.
#
# "\C-xs": "\C-e\"\C-asu -c \""
```

This would mean that if you want to add say: `Ctrl+W` to add the command `time` before another command you would do:

```
"\C-w": "\C-e\ \C-atime \
```

Another example would be, if you want to add: `Ctrl+W Q` to add: `echo "<command>"` around the `<command>` you would do:

```
"\C-wq": "\C-e\"\C-aecho \""
```

This means that if you type `word` and then press `Ctrl+W` followed by `Q` you will end up with: `echo "word"`, pretty handy.

You can also add a `.inputrc` in your home dir with the same functions, but only for that user.

Just make sure you don't overwrite some other function, test the `Ctrl+<key(s)>` function that you wanna use so they don't already do something.

If you want to bind keys to functions or characters, this is not the place to do that, then you need to find your keymap like this one:

```
/usr/lib/kbd/keymaps/i386/qwerty/se-latin1.kmap.gz
```

gunzip it, edit it and then zip it up again.

I will not explain how to edit a keymap here, but it's not that hard, just read the contents of the unzipped keymap a few times and use the power of deduction.

The `/etc/passwd` holds the login information which looks something like this:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
lp:x:4:7:lp:/var/spool/lpd:
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
alien:x:500:500:Me:/home/alien:/bin/bash
user:x:501:501::/home/user:/bin/csh
```

You are looking on 7 accounts, namely: `root`, `bin`, `daemon`, `adm`, `lp`, `shutdown`, `alien` and `user`.

Each of the lines have 7 fields separated by "`:`". The fields from left to right are:

```
1 login-name
2 encrypted-password (this field contains only 'x' if there is an /etc/shadow)
3 uid (user id)
4 gid (group id)
5 user-information
6 home-directory
7 shell-type
```

If you make an account by hand in the `/etc/passwd` file, put a '*' in the encrypted-password field and use the `passwd` command to set the password.

The `/etc/shadow` file, if this file exists, this is where the real encrypted passwords are located, this file can only be read by the super-user (root), and it looks like this:

```
root:$1$UrbUdguK$yrO3U/dlwKC5K3y2ON/YM.:11056:0:21:7:::
bin:*:11056:0:99999:7:::
daemon:*:11056:0:99999:7:::
adm:*:11056:0:99999:7:::
lp:*:11056:0:99999:7:::
shutdown:$1$hu86lnLIhnklY8ijnHui7.nn/jYg/mU:11056:1:365:14:::
alien:$1$vf3tGCFF$YRoFUgFDR8CVK6hHOwU/p0:11056:0:50:14:31::
user:$1$asd8kiLY76JNdskDkj97kMiyBujy/jD:11074:2:100:14:3::
```

(I've changed the characters in the encrypted-password, so they are not valid)

The manual page (`man 5 shadow`) tells the following about the 9 fields:

```
Login name
Encrypted password
Days since Jan 1, 1970 that password was last changed
Days before password may be changed
Days after which password must be changed
Days before password is to expire that user is warned
Days after password expires that account is disabled
Days since Jan 1, 1970 that account is disabled
A reserved field
```

If anyone knows what the last field (after the final `:`) is reserved for ... please drop me a mail.

Read the lines from the files, and compare them with what the 9 fields mean, and see if you can make out how the accounts for each user is set up.

Now the `/etc/motd,/tt> file.`

The `/etc/motd` contains whatever you want to display to the user that logs into the system, this can be a set of rules for your system, or some ascii graphics or whatever you want.

And now the `/etc/skel/` which is a dir and contains the basic files that will be given to any new user account.

Say that you add a file called, `/etc/skel/.ircrc` then all new useraccounts that are added will have a `~/.ircrc` file in their home directory.

And last the `/etc/issue` and `/etc/issue.net` file.

On most systems there is only an `/etc/issue` file that works as both `/etc/issue` and `/etc/issue.net`, the issue file holds the information or rather text that is displayed to the user just before the login prompt, usually it's the system specifications, like operating system version and things like that.

The `/etc/issue` (if there is any `/etc/issue.net`) is the issue file for the local users, and the `/etc/issue.net` is for users that logs in from a remote host.

There is a lot more in the `/etc` directory, but what I've written this far is about what you need to customize your system to your needs.

```
==============================================================================
```

7 - Networking

```
============================================================ Table Of Contents
```

Linux is known to be one of the best networking operating systems in the world, perhaps even THE best, unfortunately it's not the easiest OS to set up a good network in, but I hope that this section will make exclamation marks out of some question marks.

The first thing you need to do networking is 2 computers and network cards attached with a 'crossed' network cable, or connected via a hub, with normal network cables (doh?).

The next step is to make sure the network cards work properly.

Make sure you have the networking support compiled into the kernel, you need to have the following checked in the kernel (example is from the Linux 2.2.14 kernel, using `make menuconfig`, you can read more about how you compile/recompile your kernel in `/usr/doc/HOWTO/Kernel-HOWTO`):

```
General setup  --->
        [*] Networking support
Networking options  --->
        [*] Kernel/User netlink socket
        [*] Network firewalls
        <*> Unix domain sockets
        [*] TCP/IP networking
        [*] IP: advanced router
        [*] IP: firewalling
        [*] IP: transparent proxy support
        [*] IP: masquerading
        [*] IP: ICMP masquerading
        [*] IP: aliasing support
        <*> IP: Reverse ARP
        [*] IP: Allow large windows (not recommended if <16Mb of memory)
Network device support  --->
        [*] Network device support
        Ethernet (10 or 100Mbit)  --->
                [*] Ethernet (10 or 100Mbit)
                (In here find your network card and check it.)
Filesystems  --->
        [*] /proc filesystem support
```

Then you add this line in your `/etc/rc.d/rc.local`

```
echo "1" > /proc/sys/net/ipv4/ip_forward () at boot time
```

If you have more then one network card, you may wanna add one as Module and one hard compiled (*) into the kernel, so that the kernel knows that it's 2 network cards.

Then you need to name them `eth0` and `eth1`, this you *may* have to do even if you only have 1 network card, but it's not likely.

I have 2 network cards, one "3com 509 B" and one "3com 905" The first thing I need to do is to is to find the module name for the network cards.

Go to `/lib/modules/2.2.14/misc/` and do an `ls` (the `2.2.14/` dir will be named after whatever kernel version you have: use `uname -r` to find out)

In there I found a file called 3c59x.o (that is the one I compiled as module), then I set that as eth0, like this: I open the file `/etc/conf.modules` (or `/etc/modules.conf` depending on the kernel and system) and add:

```
alias eth0 3c59x
```

Then I know the other card is a "3com 509 B" so I go to: `/lib/modules/2.2.14/net/` and in there I find a 3c509.o, so I again add an alias in `/etc/conf.modules`:

```
alias eth1 3c509
```

Basically, you will find the network cards you added from the kernel in either `/lib/modules/2.2.14/net/` or `/lib/modules/2.2.14/misc/`, or say now that you had a Linux 2.2.15 kernel then it would be: `/lib/modules/2.2.15/net/` and `/lib/modules/2.2.15/misc/`

And remember to add the cardnames without the `.o` in the module name, as `3c509.o` will be named `3c509` as an alias in `/etc/conf.modules`.

Now you wanna add the network card so it starts at boot time and get an IP. Now you must decide what network IP it should have (either `192.168.1.*` or `10.0.0.*` in this example I've used `10.0.0.*`). Open or create a file called: `/etc/sysconfig/network-scripts/ifcfg-eth0` (if it doesn't exist when you open it, it will be created.)

In this file type the following:

```
ONBOOT=yes
DEVICE=eth0
IPADDR=10.0.0.1
NETMASK=255.255.255.0
NETWORK=10.0.0.0
BROADCAST=10.255.255.255
BOOTPROTO=none
```

Then save and quit.

`ifcfg-eth0` goes if this is your first network card, if it were your second network card on the same computer it would be `ifcfg-eth1`, and then the `DEVICE` variable assignment would say `DEVICE=eth1`.

After this you need to tell your computer what IP, network name and nick name it has. This you do in `/etc/hosts`. By default you should have this line in your `/etc/hosts`:

```
127.0.0.1       localhost       localhost.localdomain
```

Now you add your new hosts, the current computer and the other computer(s), here I have used the `10.0.0.*` IP range.

```
10.0.0.1        main.bogus.com  main
10.0.0.2        sidekick.bogus.com      sidekick
```

Note that there is a `TAB` separating each of the 3 fields (IP hostname nick).

After that it's time to set up the forwarding and everything like that using ipchains. This you do by adding the following lines to your `/etc/rc.d/rc.local`

```
/sbin/ipchains -P forward DENY
/sbin/ipchains -A forward -i eth0 -j MASQ
```

You may also wanna execute the lines since `/etc/rc.d/rc.local` only loads at boot time.

At this time you may also wanna set up a caching nameserver on your system, both to speed up your surfing and to get your LAN (Local Area Network) to interact in a proper way.

In the following example I've used:

```
bind-8.2.2P5-1mdk.i586.rpm
```

and

```
caching-nameserver-6.0-3mdk.noarch.rpm
(A nameserver is depending on bind)
```

So after you installed `bind` and a caching nameserver this is what you wanna do, (everything in this example is based on the previously written network configurations):

First you need to edit a file named `/etc/named.conf`, where in you add a "zone". The zones in this example to add, looks like this:

```
zone "bogus.com" {
        type master;
        file "bogus.com";
};

zone "0.0.10.in-addr.arpa" {
        type master;
        file "10.0.0";
};
```

The first one is for the networked computers' hostnames, and the second for their IP's.

In this example I use `10.0.0.*` as the network IP, but another common network IP is also `192.168.0.*` .... those are the two most common/accepted ones.

Then you save and quit that, to go and create the files `bogus.com` and `10.0.0`, which you do in: `/var/named/`

First we can create `/var/named/bogus.com`, and in there type the following:

```
@       IN      SOA     ns.bogus.com. main.bogus.com.  (
                                       2000020100 ; Serial
                                       28800      ; Refresh
                                       14400      ; Retry
                                       3600000    ; Expire
                                       86400 )    ; Minimum
              IN      NS      localhost.
      ;1      IN      PTR     localhost.
```

```
localhost       A       127.0.0.1
ns              A       10.0.0.1
sidekick        A       10.0.0.2
main            A       10.0.0.1
mail            A       10.0.0.1
```

(What comes before the Serial, 2000020100 is a date, 2000-02-01-00, so you can type that as your current date.)

Then you save and quit that, and create the file: `/var/named/10.0.0`, and in there you type this:

```
@       IN      SOA     ns.bogus.com. main.bogus.com.  (
                              2000020100 ; Serial
                              28800      ; Refresh
                              14400      ; Retry
                              3600000    ; Expire
                              86400 )    ; Minimum
                NS      ns.bogus.com.
1               PTR     main.bogus.com.
2               PTR     sidekick.bogus.com.
```

Now it's almost time to start the nameserver, but first you wanna add the nameserver to your `/etc/resolv.conf` so you have any use of it.

Open `/etc/resolv.conf` and at the top of it add:

```
nameserver 10.0.0.1
```

and leave the rest of the file entries as they are if there are any, then save and quit that.

And now it's time to start the nameserver. To be sure that everything works normally, do these commands:

```
/usr/sbin/named
/usr/sbin/ndc restart
```

And then type 'nslookup', that should look like this:

```
root:~# nslookup
Default Server:  main.bogus.com
Address:  10.0.0.1

>
```

If you get that, just type `exit` at the > prompt, and then add the following lines to `/etc/rc.d/rc.local`

```
if ps aux | grep named | grep -v grep >/dev/null ; then
    echo >/dev/null
else
    /usr/sbin/named
    /usr/sbin/ndc restart
fi
```

This will check if you have a name daemon `named` running, and if not, it will start it, note that this is not the 100% correct way to do it, but it's by far they most lazy way to do it, and it works.

That was the basics of making a network at home and setting up a nameserver. I hope it's enough so that anyone can set up a little network at home.

==

8 - The init and system scripts

== Table Of Contents

In this section I will cover the system V `init`, which is the most used `init` in Linux.

Beside the Syst V `init`, there are also the BSD `init`, which is used by Slackware and Debian and in some smaller distributions of Linux. The rest, as far as I know, uses the Syst V `init`. There are not so much difference of the two, I'll try to cover the differences later.

The example and files here are taken from `SysVinit-2.78-6` & `initscripts-5.27-37` which is compatible in some ways with the BSD `init`, I'll come back to that later.

So here we go:

The basic Syst V init comes with the following commands & devices:

```
/dev/initctl           This is the init control device.
/sbin/halt             This is to shut down the system.
/sbin/init             This is to change the init runlevel.
/sbin/killall5         This will kill everything but the script that runs it.
/sbin/pidof            This will get the PID of a Process name.
/sbin/poweroff         This will power down the system.
/sbin/reboot           This will reboot the system.
/sbin/runlevel         This will tell the init runlevel.
/sbin/shutdown         This will shut down the system.
/sbin/sulogin          This is the single user mode login.
/sbin/telinit          This is the init process control initialization.
/usr/bin/last          This shows who was in the system last.
/usr/bin/lastb         This is about the same as last.
/usr/bin/mesg          This is to toggle writable mode on your tty.
/usr/bin/utmpdump      This dumps a file for utmp (this lacks documentation)
/usr/bin/wall          This sends a message to all ttys.
```

And then the `init` needs the following extra files/dirs and commands from the `initscripts` package:

```
/bin/doexec            This lets you run a program under another name.
/bin/ipcalc            This lets you manipulate IP addresses.
/bin/usleep            This sleeps for microseconds.
/etc/X11/prefdm        This is the display manager preference file for X.
/etc/adjtime           This is the Kernel Clock Config file.
/etc/init.d            BSD init compatibility directory.
/etc/initlog.conf      This is the initlog configuration file.
/etc/inittab           This is the main init configuration file.
/etc/modules           This is where the kernel loads modules from at boot.
/etc/ppp/ip-down       This is a script for dialup internet connections.
/etc/ppp/ip-up         This is a script for dialup internet connections.
/etc/profile.d/inputrc.csh      Shell Key bindings for csh and tcsh.
/etc/profile.d/inputrc.sh       Shell Key bindings for sh and bash.
/etc/profile.d/lang.csh         Language files - i18n stuff for csh and tcsh.
/etc/profile.d/lang.sh          Language files - i18n stuff for sh and bash.
/etc/profile.d/tmpdir.csh       Set temporary directory for csh and tcsh.
/etc/profile.d/tmpdir.sh        Set temporary directory for sh and bash.
/etc/rc.d/init.d/functions      Functions for scripts in init.d
/etc/rc.d/init.d/halt           Runlevel 0 (shutdown/halt) script.
/etc/rc.d/init.d/kheader        Script to regenerate the /boot/kernel.h file.
/etc/rc.d/init.d/killall        Script to make sure everything is shut off.
/etc/rc.d/init.d/mandrake_everytime     Mandrake specific stuff.
/etc/rc.d/init.d/mandrake_firstime      Mandrake post install stuff.
/etc/rc.d/init.d/netfs          Mounts network filesystems.
/etc/rc.d/init.d/network        Bring up/down networking.
/etc/rc.d/init.d/random         Script to help random number generation.
/etc/rc.d/init.d/rawdevices     Device stuff for applications such as Oracle.
/etc/rc.d/init.d/single         Single user script (runlevel 1)
/etc/rc.d/init.d/sound          Launch sound.
/etc/rc.d/init.d/usb            Launch USB support.
/etc/rc.d/rc.local              Boot time script, (like autoexec.bat in DOS).
/etc/rc.d/rc.modules            Bootup script for modules.
/etc/rc.d/rc.sysinit            Main system startup script.
/etc/rc.d/rc0.d/S00killall      Runlevel 0 killall script link.
/etc/rc.d/rc0.d/S01halt         Runlevel 0 halt script link.
/etc/rc.d/rc1.d/S00single       Runlevel 1 single script link.
/etc/rc.d/rc2.d/S99local        Runlevel 2 local script link. (rc.local)
/etc/rc.d/rc3.d/S99local        Runlevel 3 local script link. (rc.local)
/etc/rc.d/rc4.d                 Runlevel 4 directory.
/etc/rc.d/rc5.d/S99local        Runlevel 5 local script link. (rc.local)
/etc/rc.d/rc6.d/S00killall      Runlevel 6 killall script link.
/etc/rc.d/rc6.d/S01reboot       Runlevel 6 reboot script link.
/etc/rc.local                   BSD init compatibility file....?
/etc/rc.sysinit                 BSD init compatibility file....?
/etc/rc0.d                      BSD init compatibility directory.
/etc/rc1.d                      BSD init compatibility directory.
/etc/rc2.d                      BSD init compatibility directory.
/etc/rc3.d                      BSD init compatibility directory.
/etc/rc4.d                      BSD init compatibility directory.
/etc/rc5.d                      BSD init compatibility directory.
/etc/rc6.d                      BSD init compatibility directory.
/etc/sysconfig/console          Directory for console stuff, like the keymap.
/etc/sysconfig/init             Basic init boot configurations.
/etc/sysconfig/network-scripts/ifcfg-lo         Network config for localhost.
/etc/sysconfig/network-scripts/ifdown           Turning off interfaces script.
/etc/sysconfig/network-scripts/ifdown-post      Post stuff for ifdown.
```

```
/etc/sysconfig/network-scripts/ifdown-ppp       Turning off ppp.
/etc/sysconfig/network-scripts/ifdown-sl        Turning off SLIP.
/etc/sysconfig/network-scripts/ifup             Turning on interfaces script.
/etc/sysconfig/network-scripts/ifup-aliases     Turning on alias interfaces.
/etc/sysconfig/network-scripts/ifup-ipx         Turning on IPX.
/etc/sysconfig/network-scripts/ifup-plip        Turning on PLIP.
/etc/sysconfig/network-scripts/ifup-post        Post stuff for ifup.
/etc/sysconfig/network-scripts/ifup-ppp         Turning on ppp.
/etc/sysconfig/network-scripts/ifup-routes      Turning on routes.
/etc/sysconfig/network-scripts/ifup-sl          Turning on SLIP.
/etc/sysconfig/network-scripts/network-functions     Functions for the scripts.
/etc/sysconfig/rawdevices       Raw device bindings.
/etc/sysctl.conf                System Control configurations.
/sbin/consoletype               This prints the console type.
/sbin/getkey                    Prints the key strokes....
/sbin/ifdown                    Application for the previous config files.
/sbin/ifup                      Application for the previous config files.
/sbin/initlog                   Logs msgs and events to the system logger.
/sbin/installkernel             Installs a kernel (not for manual use).
/sbin/minilogd                  * Totally lacking documentation.
/sbin/netreport                 Reports changes of the network interface.
/sbin/ppp-watch                 Application used by ifup-ppp.
/sbin/service                   Can send commands to all services etc.
/sbin/setsysfont                Sets the system font.
/usr/bin/listhome               Lists the users home directories.
/usr/sbin/detectloader          Detect the current boot loader.
/usr/sbin/supermount            Automatic mount/umount application.
/usr/sbin/sys-unconfig          System reconfiguration tool.
/usr/sbin/usernetctl            User Network interface control application.
/var/log/wtmp                   Previously logged in users entries.
/var/run/netreport              Directory for the netreport application.
/var/run/utmp                   Currently logged in users entries.
```

So what do you really need to know of all that?

Well, here's the simple basics of how it works and what you need to remember.

See below for an explanation of the `/etc/inittab` file.

Here is how the runlevels works:

The Runlevel can be one of `1` to `6` and the number means this:

```
0 - halt
1 - Single user mode
2 - Multiuser, without NFS
3 - Full multiuser mode
4 - Unused
5 - X11
6 - reboot
```

You change the runlevel with the `init` command, so say that you are at init runlevel 3 and you wanna go to single user mode for some reason, then you can do: `init 1`

In a single user mode you can only be one user, root.

And in a single user environment you can't do networking and other tasks, the runlevel 1 is meant to be there for system maintenance.

The two mostly used runlevels as default is 3 and 5. Mandrake and RedHat etc. uses Runlevel 5 as default, and so they start up with a GUI in X Windows.

Typing `init 0` will shut down the system, and typing runlevel `6` will reboot the system.

What determines what the various runlevels actually start at boot time is what is in their respective directory:

```
Runlevel 0: /etc/rc.d/rc0.d/
Runlevel 1: /etc/rc.d/rc1.d/
Runlevel 2: /etc/rc.d/rc2.d/
Runlevel 3: /etc/rc.d/rc3.d/
Runlevel 4: /etc/rc.d/rc4.d/
Runlevel 5: /etc/rc.d/rc5.d/
Runlevel 6: /etc/rc.d/rc6.d/
```

So, here say that you wanna stop your web server from starting at boot time. The first thing you wanna do is to find out what runlevel you are in, that you do with the runlevel command like this:

```
alien:~$ runlevel
N 3
alien:~$
```

This means that you are in runlevel `3`. So from here go to `/etc/rc.d/rc3.d/` which is the directory for runlevel 3.

```
alien:~$ cd /etc/rc.d/rc3.d/
alien: /etc/rc.d/rc3.d/ $
```

Here you find the file that starts the webserver (this file is usually called `httpd` with a character and a number in front of it (I'll explain the character and the numbers soon), so list the contents of the current directory and find it, or just do like this:

```
alien: /etc/rc.d/rc3.d/ $ ls -l *httpd
lrwxrwxrwx    1 root     root       15 Dec  5 06:14 S85httpd -> ../init.d/httpd
alien: /etc/rc.d/rc3.d/ $
```

This says that `S85httpd` is a link to `/etc/rc.d/init.d/httpd` (`../init.d/` if you're standing in `/etc/rc.d/init.d/` mean `/etc/rc.d/init.d/`)

So just remove the link like this:

```
alien: /etc/rc.d/rc3.d/ $ rm -f S85httpd
alien: /etc/rc.d/rc3.d/ $
```

And that is how you stop something from starting with the runlevel.

Now, if you rather would have something more start with the runlevel at boot time you do like this:

You make a simple script that starts what you wanna have started and put it in `/etc/rc.d/init.d/`.

Say that your script's name is `mystart`, you are in runlevel 3 and you have made your script executable (`chmod a+x mystart`), and you have it in your own home directory, then you do like this:

```
alien: ~$ cp mystart /etc/rc.d/init.d/
alien: ~$ cd /etc/rc.d/rc3.d
alien: /etc/rc.d/rc3.d/ $ ln -s ../init.d/mystart Z00mystart
alien: /etc/rc.d/rc3.d/ $
```

And that is all of that.

So now, what does the `Z00` in `Z00mystart` or `S85` in `S85httpd` mean? Well, as the system starts it will read file after file in its runlevels directory in alphabetical order, so to get them to start in a particular order, the link names are made to determine that order. So the later in the alphabet the first character is the later it will boot, and the same for the number, the higher number the later it will start.

So `A00something` will start before `A01something` and `Z99something` will start later than `X99something` and so on.

To get something to start at boot time you can also add it as a command in the `/etc/rc.d/rc.local` (or for some systems `/etc/rc.local`) file. That file is meant to be used for single commands and not to start up major things like a server etc.

Always try to load things with the actual runlevel which is the more correct way, rather than adding them to the `rc.local` file.

So what's the difference between the BSD `init` and the System V `init`? The only thing that differs them that you need to remember is that they store the startup scripts in different places.

The startup scripts for the BSD `init` is mainly in the following places:

```
/etc/rc0.d/
/etc/rc1.d/
/etc/rc2.d/
/etc/rc3.d/
/etc/rc4.d/
/etc/rc5.d/
/etc/rc6.d/
/etc/rc.boot/
/etc/rcS.d/
/etc/init.d/
```

While the Syst V `init` stores its scripts mainly in:

```
/etc/rc.d/rc0.d/
```

```
/etc/rc.d/rc1.d/
/etc/rc.d/rc2.d/
/etc/rc.d/rc3.d/
/etc/rc.d/rc4.d/
/etc/rc.d/rc5.d/
/etc/rc.d/rc6.d/
/etc/rc.d/init.d/
```

In the BSD init the `/etc/rc.boot/` and the `/etc/rcS.d/` directories are more or less substitutes for the `rc.local` file since you can put things in them that starts up at boot time... what you put in `/etc/rcS.d/` will even start at single user mode, so be careful what you put there.

So basically, the actual scripts goes in the `init.d` directory and you link them to the runlevel directory with a prefix to determine where in the bootup they should be loaded.

Here is an example of how an `init` script can be made. Here I made a script that would start a daemon named `daemon`:

```
#!/bin/sh
# example       Example init script that would load 'daemon'
#
# Version:      @(#) /etc/rc.d/inet.d/example 0.01 19-Feb-2001
#
# Author:       Billy (Alien), <alien a ktv d koping d se>
#

. /etc/rc.d/init.d/functions

function status() {
    ps aux | grep daemon &&
    echo "Daemon is running." ||
    echo "Daemon is not running."
}

case "$1" in
    start)
        # Check if daemon is in our path.
        if `which daemon` > /dev/null; then success || failure; fi
        echo -n "Starting Daemon"
        daemon
        echo
        ;;
    stop)
        # Check if daemon is in our path again.
        if `which daemon` > /dev/null; then success || failure; fi
        echo "Stopping Daemon"
        killall -15 daemon
        ;;
    status)
        echo "Status of Daemon:"
        status
        ;;
    reload)
        echo "Restarting Daemon."
        killall -1 daemon
        ;;
    restart)
        if `which echo` > /dev/null; then success || failure; fi
        $0 stop
        $0 start
        ;;
    *)
        echo "Usage: $0 start|stop|restart|status"
        exit 0
esac
```

A note is that the success and failure functions/commands come from the `/etc/rc.d/init.d/functions` file, which may not be present in all distributions of Linux, since it as far as I know only comes with RedHat and Mandrake.

The `init`'s main configuration file is the `/etc/inittab` file, here is where you set which runlevel you wanna have, and how many consoles you want etc, so here we go:

The line where you actually set the runlevel looks like this (here runlevel 3):

```
id:3:initdefault:
```

Most RedHat like systems have runlevel 3 or 5 as default, but if you don't have any networking, you may find it better to change to runlevel 2.

Next in this file should be the system initialization.

```
si::sysinit:/etc/rc.d/rc.sysinit
```

This line tells the system the path to the `rc.sysinit` where it loads a lot in the system, as system clock, sets the hostname, and performs a number of checks.

Next in line is this:

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
```

This tells the system where to load the programs and daemons it should load for the runlevel it's in.

Say that we are in runlevel 3 (Default) then it looks at this line:

```
l3:3:wait:/etc/rc.d/rc 3
```

And there after goes to load all what's in `/etc/rc.d/rc3.d/` (`rc3.d` or any `rc?.d` contains links to the real files or scripts that is located in `/etc/rc.d/init.d`, so if you wanna add something to your runlevel, just look how they have done it and do it in a similar fashion. and make sure to not start any network dependent application before the network starts and so on...)

Then it comes some other various stuff as trap the `Ctrl+Alt+Del` etc.

After this comes the tty's (Terminal Types), and there locations.

```
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

If you wanna add some more you can add like this:

```
8:2345:respawn:/sbin/mingetty tty8
9:2345:respawn:/sbin/mingetty tty9
10:2345:respawn:/sbin/mingetty tty10
11:2345:respawn:/sbin/mingetty tty11
```

And leave `tty7` reserved for X-windows.

Last in the file should only be some line about `xdm` and its location, that is, if you have `xdm` installed...

And if you have read the tutorial to this point you shouldn't need any real explanation of this script.

If you still don't understand how the init scripts work, read the scripts in your system and try to understand them. And also read this section about the `init` and the init scripts again.

==

9 - Other Frequently asked questions with answers

== Table Of Contents

Q: How can I play an `.au` file?

A: The file suffix `.au` means it's an audio file, so you can either do the same as for mp3's: `mpg123 file.au` or you can (since `.au` files are raw audio):

```
cat file.au > /dev/dsp
```

or:

```
cat file.au > /dev/audio
```

Q: What exactly is bash?
A: If you have read this tutorial you would know ... it's a command shell type.

Q: What's a [command] shell?
A: A command shell provides the user interface around the operating system: since you can't get the O.S. to do anything directly, you need a command shell which accepts commands, interacts with the O.S. and performs actions for you.

Q: I wanna make some sub-directory's to `/usr/local/` but I'm too lazy to write almost the same line over and over again, is there any easier way?
A: Yes there is: `mkdir /usr/local/{dir1,dir2,dir3}`

Q: I have a dir with files like this: `file.1`, `file.2`, `file.3`, etc. Is there any way I can list say, `file.3`, `file.4`, `file.6`, `file.9` only without using grep?
A: Yes there is: `ls file*[3469]*`

Q: I want to remove some files with wildcards included in the command (`rm -rf .??*`), is there any way I can see what the command will remove before I actually do the command?
A: Yes there is: `echo rm -rf .??*`

Q: How can I remove file names that are/have special characters like a file named: `-!?* !!?`
A: Do this: `rm ./-\!\?*\ \!\!\?`
The `./` makes sure it will look in the current directory, and the `\` (back slashes) will make sure that the special characters special meaning is over looked, so they are treated just like any other character.

Q: I accidentally deleted something on my system, is there any way of getting it back?
A: Yes there is, read this file `/usr/doc/HOWTO/mini/Ext2fs-Undeletion`
(or `/usr/doc/HOWTO/mini/Ext2fs-Undeletion.gz`)
This is also good knowledge if someone *hacks* your system and thinks they are safe just because they deleted the logs. Or you can download a program called 'recover' from www.freshmeat.net

Q: How do I compile my kernel?
A: Read this file: `/usr/doc/HOWTO/Kernel-HOWTO`

Q: Can I make a variable read-only so no one can change it?
A: Yes you can: `readonly variable_name`
As say that I have a variable: `$myvar`
Then I would do: `readonly myvar`
And to display all the read-only variables just do: `readonly`

Q: How can I display all the set variables?
A: With the command: `set`
To get the environment use this command: `env`
And to display the system variables, type a `$` and press `TAB`.

Q: Something is wrong with a script, but I don't know what, how can I find out?
A: Turn on command echoing by doing this: `/bin/sh -x <scriptname>`
`+` means that the command was successful.
`-` means that the command was unsuccessful.

Q: I'm using `hexedit` and similar commands a lot, but I'm getting too lazy to find out where all the binaries live, isn't there any faster way of opening a binary or script that is in my `$PATH` than to locate, find or something like that and then go from there?
A: Well, I don't know how much faster it is, but you can always do this: ``hexedit `which <file>` ``

Q: How can I list only the directories in a directory without the files?
A: Well, some distros has an alias `lsd`, and it does: `ls -d */`

Q: How can I transfer a file if the computer doesn't have any FTP?
A: If you wanna actually copy a file to a place that don't have ftp or for some other reason you can't use ftp, there is 2 commands that can do this: `scp` (secure copy - requires `ssh`) and `rcp` (remote copy - requires `rlogin`)
They work like this:

```
scp local.file user@host.domain.com:/remote/directory/
rcp local.file user@host.domain.com:/remote/directory/
```

Q: Is there any way I can view my processes so I can see the free stack, father processes siblings and children etc.?
A: Yes there is, press: `Ctrl + Scroll-Lock`

Q: Is there any way I can view the memory buffers etc.?
A: Yes there is, press: `Shift + Scroll-Lock`

Q: Is there any way to get some info from the stack?
A: Yes there is press: `AltGr + Scroll-Lock`

Q: My sound volume is really low when I play stuff, how can I change it?
A: Use the program `aumix`. It should be on most systems by default, and it's pretty self explanatory.

Q: I think it's a pain pressing up arrow until I reach the command I want to use, that is too long to type again, is there any faster way?
A: Yes, you can doing a search by pressing `Ctrl+R` and type something that matches a previously used command; edit it if you need to, and just press enter to execute it or `Ctrl+C` to cancel.

Q: Is there any console based mpeg movie players?
A: Not that I know of, but if there is, I'd like to know myself.

Q: How can I take a screenshot without having to install Gimp or something like that?
A: Well, the easiest way of taking a screen shot in X is to do the following command:

```
import -window root foo.jpg
```

That will dump a screen shot as `foo.jpg`, in your current directory.

Q: How can I import variables from one script to another, like a config file?
A: You can read all variables from a file by doing `. <file name>`; say that you have a file called `myvars` and you want to you those variables in a script, then you add the line:

```
. myvars
```

in the beginning of the script, after the `#!/bin/bash` line. The "`.`" is actually a command, that reads and executes commands in a file, which here works to imported the variables since they are executed from within the script.

Q: I'm using `'port sentry'` as a firewall control software, now my routing table is over full, how can I take away all that is routed to `localhost` in one command or string?
A: Well like this:

```
for ips in `route | grep local | cut -c 1-14`; do route del -host $ips gw 127.0.0.1 2>/dev/null ; done
```

That should do the trick.

Q: If I'm in a directory full of `.rpm` files and I wanna find out which rpm that contains, say, the file `vga.h`, how would I do that?
A: Well you can do this:

```
for foo in `ls -1 *.rpm`; do rpm -qlp $foo | grep vga.h 1>/dev/null 2>&1 && echo $foo; done
```

Q: Is there any place I can find out where all the `^` (`Ctrl`) characters mean?
A: Yes, look in the ascii table below, it works like this:
M is the 13'th character in the alphabet, and in the ascii table 013 (dec) has the value CR which means Carriage Return. This means that `^M` (`Ctrl+M`) is the same as pressing the return button.
And `^A` which is 001 (dec) is Start of Header, in other words take the cursor to the beginning of the current line.

Q: How do I write stuff in hex code?
A: Here are some general things you can do with hex codes. Start by referring to this ASCII table:

```
ASCII Codes (7-bit)

Decimal   Octal  Hex    Binary      Value
-------   -----  ---    ------      -----
  000      000    00   00000000      NUL    (Null char.)
  001      001    01   00000001      SOH    (Start of Header)
  002      002    02   00000010      STX    (Start of Text)
  003      003    03   00000011      ETX    (End of Text)
  004      004    04   00000100      EOT    (End of Transmission)
  005      005    05   00000101      ENQ    (Enquiry)
  006      006    06   00000110      ACK    (Acknowledgment)
  007      007    07   00000111      BEL    (Bell)
  008      010    08   00001000       BS    (Backspace)
  009      011    09   00001001       HT    (Horizontal Tab)
  010      012    0A   00001010       LF    (Line Feed)
  011      013    0B   00001011       VT    (Vertical Tab)
  012      014    0C   00001100       FF    (Form Feed)
  013      015    0D   00001101       CR    (Carriage Return)
  014      016    0E   00001110       SO    (Serial In)
  015      017    0F   00001111       SI    (Serial Out)
  016      020    10   00010000      DLE    (Data Link Escape)
  017      021    11   00010001      DC1 (XON) (Device Control 1)
  018      022    12   00010010      DC2       (Device Control 2)
  019      023    13   00010011      DC3 (XOFF)(Device Control 3)
  020      024    14   00010100      DC4       (Device Control 4)
  021      025    15   00010101      NAK    (Negative Acknowledgement)
  022      026    16   00010110      SYN    (Synchronous Idle)
  023      027    17   00010111      ETB    (End of Trans. Block)
  024      030    18   00011000      CAN    (Cancel)
  025      031    19   00011001       EM
  026      032    1A   00011010      SUB
```

027	033	1B	00011011	ESC	(Escape)
028	034	1C	00011100	FS	(File Separator)
029	035	1D	00011101	GS	
030	036	1E	00011110	RS	(Request to Send)
031	037	1F	00011111	US	
032	040	20	00100000	SP	(Space)
033	041	21	00100001	!	
034	042	22	00100010	"	
035	043	23	00100011	#	
036	044	24	00100100	$	
037	045	25	00100101	%	
038	046	26	00100110	&	
039	047	27	00100111	'	
040	050	28	00101000	(	
041	051	29	00101001	)	
042	052	2A	00101010	*	
043	053	2B	00101011	+	
044	054	2C	00101100	,	
045	055	2D	00101101	-	
046	056	2E	00101110	.	
047	057	2F	00101111	/	
048	060	30	00110000	0	
049	061	31	00110001	1	
050	062	32	00110010	2	
051	063	33	00110011	3	
052	064	34	00110100	4	
053	065	35	00110101	5	
054	066	36	00110110	6	
055	067	37	00110111	7	
056	070	38	00111000	8	
057	071	39	00111001	9	
058	072	3A	00111010	:	
059	073	3B	00111011	;	
060	074	3C	00111100	<	
061	075	3D	00111101	=	
062	076	3E	00111110	>	
063	077	3F	00111111	?	
064	100	40	01000000	@	
065	101	41	01000001	A	
066	102	42	01000010	B	
067	103	43	01000011	C	
068	104	44	01000100	D	
069	105	45	01000101	E	
070	106	46	01000110	F	
071	107	47	01000111	G	
072	110	48	01001000	H	
073	111	49	01001001	I	
074	112	4A	01001010	J	
075	113	4B	01001011	K	
076	114	4C	01001100	L	
077	115	4D	01001101	M	
078	116	4E	01001110	N	
079	117	4F	01001111	O	
080	120	50	01010000	P	
081	121	51	01010001	Q	
082	122	52	01010010	R	
083	123	53	01010011	S	
084	124	54	01010100	T	
085	125	55	01010101	U	
086	126	56	01010110	V	
087	127	57	01011111	W	
088	130	58	01011000	X	
089	131	59	01011001	Y	
090	132	5A	01011010	Z	
091	133	5B	01011011	[	
092	134	5C	01011100	\	
093	135	5D	01011101	]	
094	136	5E	01011110	^	
095	137	5F	01011111	_	
096	140	60	01100000	`	
097	141	61	01100001	a	
098	142	62	01100010	b	
099	143	63	01100011	c	
100	144	64	01100100	d	
101	145	65	01100101	e	
102	146	66	01100110	f	
103	147	67	01100111	g	
104	150	68	01101000	h	
105	151	69	01101001	i	
106	152	6A	01101010	j	
107	153	6B	01101011	k	
108	154	6C	01101100	l	

```
   109      155    6D   01101101       m
   110      156    6E   01101110       n
   111      157    6F   01101111       o
   112      160    70   01110000       p
   113      161    71   01110001       q
   114      162    72   01110010       r
   115      163    73   01110011       s
   116      164    74   01110100       t
   117      165    75   01110101       u
   118      166    76   01110110       v
   119      167    77   01110111       w
   120      170    78   01111000       x
   121      171    79   01111001       y
   122      172    7A   01111010       z
   123      173    7B   01111011       {
   124      174    7C   01111100       |
   125      175    7D   01111101       }
   126      176    7E   01111110       ~
   127      177    7F   01111111      DEL
```

Say that you want to echo `Hi` with hex code, you do this:

```
echo -e "\x048\x069"
```

The `\x` part is to let `echo -e` know that it's hexadecimal code. You can even hide commands like that.

Here is a script example of hiding the `top` command in hex and execute it:

```
#!/bin/bash

hexcode='\x074\x06F\x070'

`echo -e $hexcode`
```

This will execute what `echo` echos, due to the ``` ` ```'s. So that script will actually start the `top` command.

Q: Another question I got a while back is how to make a `DWORD` (Double Word), that means how to rewrite an address or IP to hex/oct/dec. And it's not that hard, all it takes is some mathematics. It works like this. There are several methods, but let's start with the Decimal way of making a `DWORD`.

Say you have IP: 127.0.0.1, then you do:

```
127 * 16777216 = 2130706432
0 * 65536      = 0
0 * 256        = 0
1 * 1          = 1
Sum:             2130706433
```

Or if you have the IP: 123.123.123.123

```
123 * 16777216 = 2063597568
123 * 65536    = 8060928
123 * 256      = 31488
123 * 1        = 123
Sum:             2071690107

Note:
      16777216 = 2^24
         65536 = 2^16
           256 = 2^8
             1 = 2^0
```

Next method is to convert it to HEX, OCT etc, to convert from dec to oct etc. you can either use the ascii table a few lines up, or you can download a program called ascii from http://www.freshmeat.net. Last time I saw it, it was located at: http://freshmeat.net/projects/ascii/download/ascii-3.0.tar.gz.

So assuming you have downloaded that and wanna covert `127.0.0.1` and `123.123.123.123` to hex `DWORD`s, do this:

```
alien:~$ ascii 127 0 0 1
ASCII 7/15 is decimal 127, hex 7f, octal 177, bits 01111111: called ^?, DEL
Official name: Delete

ASCII 5/7 is decimal 087, hex 57, octal 127, bits 01010111: prints as `W'
Official name: Majuscule W
Other names: Capital W, Uppercase W

ASCII 3/0 is decimal 048, hex 30, octal 060, bits 00110000: prints as `0'
Official name: Digit Zero
```

```
ASCII 3/0 is decimal 048, hex 30, octal 060, bits 00110000: prints as `0'
Official name: Digit Zero

ASCII 3/1 is decimal 049, hex 31, octal 061, bits 00110001: prints as `1'
Official name: Digit One

alien:~$
```

Take the hex numbers after the decimal of 127, 0 and 1 and you'll come to the conclusion that `127.0.0.1` in a hex `DWORD` is `7F000001`.

You can use OCT the same way .. with as many leading `0`'s as you please:

```
0173.0173.0173.0173 or 000173.00000173.000173.0000000000000000000000173,
```

it still means IP `123.123.123.123`.

And of course to add on the confusion you can mix the methods.

```
0173.0x7b.00173.123
```

Now there is even more to this like that you can add any multiple of the number 4294967296 (2^32) to the number without the IP changing .... But let's not get into that ....

So basically typing:

```
http://0173.0x7b.00173.123/
```

in your web browser will end you up at IP `123.123.123.123` (which doesn't exist) but the idea is the same for everything, so if you see some lame spammer thinking that you won't know from what address he sent something ... The just back count it and send abuse mail to his internet service provider.

If someone has more questions mail them to me at: alien a ktv d koping d se maybe I'll include them in the tutorial, but I'll do my best to answer the questions anyway.

```
===============================================================================
```

10 - Basics of the common UNIX and Linux text editors

=== Table Of Contents

Here we go with the text editors `vi`, `ed` and `emacs`. `ed` is just explained for historical reasons.

Most commonly used VI commands

Here we go with the `vi` commands, these are illogical but still good to know because all computers don't have `emacs`, `joe`, `pico` and so on. Solaris / SunOS comes default with `vi` as only text editor. `vi` has 2 basic modes, command mode and edit mode, you change between them by pressing the `Esc` button, and to start to edit a file you must have a free line, which you get by pressing `Esc` followed by `o`. `vi` is bound to be the hardest and most confusing text editor to learn, and it has LOTS of commands, I included just a few of the most used commands.

> AL: the `vi` clone `vim` (& `gvim`) supplement the 'old' commands, such as `h`, `j`, `k`, `l`, with 'normal' PC keystrokes, such as up-arrow, etc. Another clone of `vi` is the stripped-down version, with only a tiny subset of available commands and functionality, in `busybox`, which is typically used in embedded systems or other limited-memory situations.

So here we go with the `vi` commands:

Inserting text

```
esc + i         insert text informant of the cursor
esc + a         append text after the existing text
esc + A         move to the end of the current line and append text
esc + O         opens new line above the current line
esc + o         opens new line under current line (insert mode)
```

Deleting text

```
esc + x         deletes one character
esc + 5x        deletes five charters
```

```
esc + dw        deletes a word
esc + 5dw       deletes five words
esc + dd        deletes the whole line
esc + D         deletes the line from cursor and forward
esc + d)        deletes the sentence from cursor and forward
esc + d(        deletes the sentence from cursor and backwards
esc + u         undelete
```

Note: `Esc + d)` or `d(` removes the sentence from cursor and forward/backwards until it reaches a dot "`.`"

Moving around in VI:

Make sure you are in command mode and the following letters will do:

```
j               moves you down
k               moves you up
h               moves you left
l               moves you right
```

Finding Text

Hit `Esc` then type in a `/` you then go to the bottom of the screen where you will see your `/`. Type in the text to look for, e.g., `/Linux` that will find the word `Linux` in the open file.

Replacing Text

Hit `Esc` and do: `:start,stop,s/s_text/r_text/g`

```
:               indicates that this is an ex command
start           is the starting line number
stop            is the stopping point
s               is the substitute command
s_text          is the search string (the text you are looking for )
r_text          is the text you are replacing with
g               is global
```

Example:

```
Esc + :5,8,s/l/ll/g
```

This would replace all `l`'s with `ll` on lines 5 to 8.

Note to Replacing Text:
Line numbers can also be:

```
.               current line
$               last line
```

Basic save & quit commands

Hit `Esc` and do a `:` where after you can type the commands.

```
w               write (save)
q               quit
!               force

ie. :q!  or  :wq
```

To create control characters do:

```
Ctrl+V Ctrl+<the character>
```

Example:

```
Ctrl+V Ctrl+A
```

That will create a `^A` character.

(These last 3 commands are very alike `ed` commands)

Another useful thing in VI is split-screen mode, so you can edit 2 files at once, this is:

```
:split
```

Just press `Esc` and type `:split`.

You can do this in most big editors ..... but of course in another way, you'll see when you're reading the emacs section.

Most commonly used ED commands

`ed` is a very very old line editor, and the grandfather of most editors, perhaps even the grandfather of all editors, it dates back to the time of the old CP/M machines, and is the father of the old DOS `edlin` line editor. So out of historical perspective, it can be fun to know how to operate `ed`.

Creating a file in `ed`:

```
alien:~$ ed newfile
newfile: No such file or directory
```

Don't worry, as soon as you save it, it will create it. `ed` is pretty simple, here's an example (the `(ed says)` and `(we type)` is just there to make it easier to follow the editing in this tutorial and is not there in reality):

```
alien:~$ echo "abcd" >> newfile; echo "efgh" >> newfile; echo "ijkl" >> newfile
alien:~$ ed newfile
15              (ed says)
1,$ n             (we type)
1       abcd    (ed says)
2       efgh    (ed says)
3       ijkl    (ed says)
1                 (we type)
abcd            (ed says)
s/ab/ll           (we type)
llcd            (ed says)
$ n               (we type)
3       ijkl    (ed says)
a                 (we type)
here we end       (we type)
.                 (we type)
w                 (we type)
27              (ed says)
q                 (we type)
alien:~$
```

Not all that hard is it?

Here's a list of the most basic commands for `ed`:

```
1,$ n           displays all lines with numbers
$ n             display last line, with number
2 n             takes you to line 2
s/new/old       replaces old with new
a               takes you to editor mode
.               takes you to command mode
d               deletes line
w               write file (save)
q               quit
```

Most commonly used Emacs commands

The final thing in this tutorial is a really quick look at some `emacs` commands:

```
^A              Cursor to the beginning of line
^E              Cursor to the end of line
^K              Deletes rest of line forward
^D              Deletes current character
^L              Horizontally center the current line
^S              Search for a word forward in the file
^R              Search for a word backwards in file

^Q              Followed by Ctrl+<anything>, gives the real control character in a text file

^X ^F           Open file
^X ^- ^-        Undo
^C+Shift+-      Undo
^X ^C !         Quit without saving
^X 2            split screen (horizontally)
^X 3            split screen (vertically)
```

```
^X O            move to other screen (if in splitscreen mode)
^X ^W           Save As
^X ^S           Save

(^X 1 to get back a single window from splitscreen mode)

Shift+Esc Shift+5: Replace query (press y to replace words)
Meta+backspace: Deletes rest of word backwards (note "Meta" == "Alt")
Home:           Takes cursor to the top of the file (Or equal to ^A)
End:            Takes cursor to the end of the file (Or equal to ^E)
Delete:         Deletes current character
Page Up / Page Down: Does what they say
```

Meta+X: Will load any emacs plugin, you may type any plugin name after pressing the Meta+X (`Alt+X`), if you press `Tab` here once you will get a list of the commands, if you type `a` followed by a `Tab` you will get all commands starting with `a` and so on .... try: Meta+X doctor to try the interactive eliza bot, or try telnet, ftp, webjump or shell.

To reach the menus "Buffers Files ..." etc. press F10 and if you wanna get out of the menus press ^G.

You may think that all this is weird, but know that emacs use to work as a VERY primitive window manager, before the times of X.

Backspace and the arrow key's works as normal.

A tip is tp press: `^X` then press `2` then press `^X` and then `O`, now press Meta+X (`Alt+X`) and type `shell`, and you should have a split window, with a shell in the lower one, so you can code or write in the upper one at the same time as you have a shell in the lower one.
To change between the windows simply press: `Ctrl+X` and then press: `o`
A note is that if you want to run `BitchX` in the shell part you need to start it with: `BitchX -d`, to get it in dumb terminal mode.

Useless or obsolete commands:

```
^I              TAB
^O              Move text forward
^P              same as UpArrow
^F              same as RightArrow
^J              Enter/Return
^B              same as LeftArrow
^N              same as DownArrow
^M              Enter/Return

^U Del          deletes 4 characters backwards
^U ^U Del       deletes 16 characters backwards
^U ^U ^U Del    deletes 64 characters backwards
```

All you really need to know to start using emacs is how to save and quit.

(`^X ^S ^X ^C` will save and quit, a tip is: hold down `^` (Ctrl) and press `X S X C`)

==

11 - Closing

=== Table Of Contents

This should be enough for you to start to script in `bash`, and make useful scripts.
The only thing that limits what you can do is your imagination (well almost).

Go over this tutorial several times so you really understand everything. If you accomplish that, you have a really good chance of learning UNIX well.

And that is what it's all about, to learn new things and explore new ways. As long as you learn you live, not the contrary.

This tutorial turned out rather large, but I hope that those of you out there that have the determination to learn shell scripting, also have the patience to read it all, and if not, you can always use it as a small dictionary.

I've got the question many times, which Linux distribution is the best .... The question in it self is pointless and as illiterate as asking what version linux is up to....

The later question can only be answered with a kernel version number, and that is what Linux is, Linux is the kernel and all distributions use the same kernel, everything else in the system is just "stuff around the kernel", to this point I've found that Mandrake is the distribution that is most compleat for my needs, and it's suitable for beginners as well as for professionals, and it has nice configuration tools that have been written especially for Mandrake.

But as I said, a Linux is a Linux, and the main difference between different distributions is the package manager, where of `rpm` is the most

standard and accepted, though I find Debian's `dpkg` good as well.

This is to the difference of distributions that have no indigenous package manager like Slackware, that emulates a package manager with its `.tgz` package format (note that `.tar.gz` is not `.tgz` since `.tgz` should have its packages compressed with their path beginning from `/`.)

Now there is nothing wrong with that if you like to compile most stuff on the system your self, and many people prefer to do that.

My conclusion is that the best distribution is the one you personally like the best, the one that fits _your_ needs.

So anyway, when you know bash scripting well enough, my suggestion is to learn C programming, which, if you look at it with bash behind you, isn't that hard.

So, I better go to bed and stop this nonsense now.

Happy scripting all of you out there.

```
================================================================================
------------- Written by Billy Wideling <-> alien a koping d net --------------
================================================================================
```

==

Appendix A - Annotated Basic Linux/UNIX commands

== Table Of Contents

This section is about Linux and UNIX basic commands and operations, and some other explanations and tricks, since this is not a command bible, I'll explain each command briefly, with a lot of help from the `man` pages and the `--help` argument (let's all thank the maker for cut & paste). Then again, I've seen files that have claimed to be UNIX command bibles that are even briefer and hold less commands... though most of the authors of those seems to be totally incapable of handling a UNIX and can't even spell, one of the worst examples I've seen was something like this: "The UNIX bible, in this phile is all the UNIX commandz j00 need" and after that was a list of commands without arguments... needless to say is also that 99% of all UNIX commands were missing. Anyway, enough of me making fun of those people now, and on with the tutorial. (Which isn't a UNIX command bible, just a note) I will refer to "*nix" here, and that means any sort of UNIX system, Linux, BSD, Solaris, SunOS, Xenix and so on included.

Index of Annotated Commands

adduser	dd	g++	lastlog	mkdir	portmap	strip	uptime
alias	declare	gcc	ldconfig	mknod	ps	su	useradd
apropos	depmod	gdb	ldd	modprobe	pstree	swapoff	userdel
awk	df	gpm	less	more	pwd	swapon	users
basename	dhcpcd	grep	lilo	mount	quota	tail	usleep
bc	dialog	halt	ln	mv	quotaoff	talk	w
BitchX	diff	hdparm	lndir	nc/netcat	quotaon	tar	wall
bzcat	dir	head	loadkeys	ncftp	quotastats	tcpdump	wc
bzip2	dmesg	help	locate	ncftpget	read	telnet	whatis
cat	do	hexdump	logout	ncftpput	reboot	test	whereis
cc	domainname	hexedit	lpq	netstat	reset	touch	which
cd	du	hostname	lpr	nice	rlogin	tr	who
chattr	echo	id	lprm	nmap	rm	traceroute	whoami
chmod	eject	ifdown	ls	ntpdate	rmmod	ulimit	whois
chown	else	ifup	lsattr	ntsys/ntsysv	route	umount	yes
chroot	env	init	lsmod	objdump	rpm	unalias	zip
cmp	exit	insmod	lsof	passwd	sed	uname	
cp	expr	install	lynx	patch	setleds	uncompress	
crontab	fdisk	ipchains	mail	pidof	seq	uniq	
cut	file	ispell	man	ping	sleep	unset	
date	find	kill	mc	pmake	sort	unzip	
dc	ftpwho	killall	mesg	pnpdump	ssh	updatedb	

Index of Annotated Commands Table Of Contents

```
adduser
    Syntax: adduser [arguments] <user>
    And can be used with the following arguments:
    -u uid
    -g group
    -G group,...
    -d home directory
    -s shell
```

```
    -c comment
    -k template
    -f inactive
    -e expire mm/dd/yy
    -p passwd
    Then there are a few arguments with no explanation:
    -o, -m, -n, and -r
    So say that you wanna add a user named "user" with password "resu"
    belonging to the group root with / as home directory using /bin/tcsh
    as shell, that would look as this:

    adduser -p resu -g root -d / -s /bin/tcsh user
```

Index of Annotated Commands Table Of Contents

```
alias
    The alias command set's an alias, as this: alias du='du -h'
    This means that whenever you type: du
    it will really do: du -h

    Typing alias by it self will display all set aliases.

    For more information on the alias command do: help alias
```

Index of Annotated Commands Table Of Contents

```
apropos
    apropos checks for strings in the whatis database. say that you
    are looking for a manual page about the `shutdown` command.
    Then you can do: apropos shutdown
    for more information, do: man whatis
    Or: man apropos
```

Index of Annotated Commands Table Of Contents

```
awk
    awk is a text formatting tool, that is HUGE, it's actually a whole
    language, some say it's not totally wrong to say that awk is not
    far off from a scripting version of C.
    However I wouldn't go as far as to say that there resemblance
    is that great.
    awk's most common use is about the same as 'cut', and it works like
    this: awk [argument-1] [argument-2] ....
    Here's some example's of converting an URL:

    echo "http://www.bogus.com/one/two.htm" | awk -F'/' '{print $3}'
    This will return: www.bogus.com
    The -F will set a delimiter, and the '{print $3}' will print the
    third field, separated by the delimiter, which is www.bogus.com,
    because there is 2 slashes, which makes the second slash the second
    field, and so www.bogus.com is the third field.
    Here's another example:

    echo "http://www.bogus.com/one/two.htm" | awk -F'/' '{print $(NF)}'
    This will return: two.htm
    The -F set's the delimiter, which once again is /, but this time
    we have used $NF which always resembles the last field.
    Another example with NF is this:

    echo "http://www.bogus.com/one/two.htm" | awk -F'/' '{print $(NF - 1)}'
    This will return: one
    Because $(NF - 1) means the last field minus one field, which always
    will be the next last field.

    You only have to use the ()'s around variables when you do something
    with them like here "$(NF - 1)", but you can use $(var) all the time
    if you want.

    Here's another example:

echo "http://www.bogus.com/one/two.htm" | awk -F'/' '{print $3 "/" $(NF - 1)}'

    This will return: www.bogus.com/one
    It will first print out the third field separated by /'s, which is
    www.bogus.com, then it will print a /, and then it will print out
    the next last field which is one.

    Here is a final example of awk:
```

```
echo "http://www.bogus.com/one/two.htm" | awk '{ while  ( $(1) ) print }'

    This will return: "http://www.bogus.com/one/two.htm" forever.
    The "while ( $(1) )" means that as long as there is first field,
    it will print the line line.
    And since there will always be a first field it will continue
    forever.
    while in awk works as this: while ( condition ) action

    As I said, awk is huge and is actually a whole language, so
    to explain all of it, it would need a tutorial of its own.
    So I will not go any deeper into awk here, but you can as always
    read its manual page which is quite large.

    So, for more info do: man awk
```

Index of Annotated Commands Table Of Contents

```
basename
    basename will strip directory and suffix from filenames.
    This command only have the two following flags:

    --help      display this help and exit
    --version   output version information and exit

    It works like this:
    alien:~$ basename /usr/local/bin/BitchX -a
    BitchX
    alien:~$ basename http://www.domain.com/path/to/file.html
    file.html
    alien:~$

    For more info do: man basename
```

Index of Annotated Commands Table Of Contents

```
bc
    A precision calculator, can be used with the following arguments:
       -l     Define the standard math library.
       -w     Give warnings for extensions to POSIX bc.
       -s     Process exactly the POSIX bc language.
       -q     Do not print the normal GNU bc welcome.
       -v     Print the version number and copyright and quit.
```

Index of Annotated Commands Table Of Contents

```
BitchX
    BitchX is usually not default on any system, but it's the far
    most advanced IRC client to *nix.
    BitchX started as a script to ircii (ircii is irc2 an extended
    irc protocol, also EPIC which is more bareboned then BitchX is
    made from ircii), until BitchX got hard coded to the protocol
    in C, by panasync I believe.

    BitchX has a lot of arguments but can be executed without any
    arguments.
    This is the syntax: BitchX [arguments] <nickname> <server list>
    And here are the arguments anyway:

    -H <hostname>   this is if you have a virtual host.
    -c <#channel>   auto join a channel, use a \ in front of the #
    -b              load .bitchxrc or .ircrc after connecting to a server
    -p <port>       connect on port (default is 6667)
    -f              your terminal uses flow controls (^S/^Q),
                    so BitchX shouldn't
    -F              your terminal doesn't use flow control (default)
    -d              dumb terminal mode (no ncurses)
    -q              don't load the rc file ~/.ircrc
    -r <file>       loads file as list of servers to connect to
    -n <nickname>   set the nickname to use
    -a              adds default servers and command line servers
                    to server list
    -x              runs BitchX in "debug" mode
    -Z              use NAT address when doing dcc
    -P              toggle check pid.nickname for running program
    -v              show client version
    -B              fork BitchX and return you to shell. pid check on.
    -l <file>       loads <file> in place of your ~/.ircrc
    -L <file>       loads <file> in place of your .ircrc and
```

```
                expands $ expands


    The most common way of starting BitchX is this, say that you want
    to have the nick 'Bash' on server irc.bogus.com, then you can do:

    BitchX Bash irc.bogus.com

    There is so much to say about BitchX that it would need a tutorial
    of its own, I'm currently writing a BitchX script, so maybe
    I'll write a BitchX tutorial some time =)
```

Index of Annotated Commands Table Of Contents

```
bzcat

    bzcat will uncompress a .bz2 file 'on the fly' as it cat's it.
    the actual file will remain compressed after bzcat has displayed
    the contents.
    bzcat has to my knowledge only one switch, and that is
    -s, that uses less memory.
    bzcat works like this:

    bzcat file.bz2

    This can be good if you wanna search something in a text file
    that has been bzip2'd.
    Examples:

    bzcat file.bz2 | grep 'text string'
    bzcat file.bz2 | wc -l
```

Index of Annotated Commands Table Of Contents

```
bzip2

    Compression tool, compresses harder then the standard gzip.
    bzip2 can be used with the following arguments:
    -h --help           print this message
    -d --decompress     force decompression
    -z --compress       force compression
    -k --keep           keep (don't delete) input files
    -f --force          overwrite existing output files
    -t --test           test compressed file integrity
    -c --stdout         output to standard out
    -q --quiet          suppress noncritical error messages
    -v --verbose        be verbose (a 2nd -v gives more)
    -L --license        display software version & license
    -V --version        display software version & license
    -s --small          use less memory (at most 2500k)
    -1 .. -9            set block size to 100k .. 900k
    Normally used as: bzip2 -d file.bz2 (to decompress a file)
    or bzip2 -z file (to compress a file)
```

Index of Annotated Commands Table Of Contents

```
cat
    cat followed by a filename will bring the contents of the file
    out to the screen (stdout), and can be used with the following
    arguments:
    -A, --show-all           equivalent to -vET
    -b, --number-nonblank    number nonblank output lines
    -e                       equivalent to -vE
    -E, --show-ends          display $ at end of each line
    -n, --number             number all output lines
    -s, --squeeze-blank      never more than one single blank line
    -t                       equivalent to -vT
    -T, --show-tabs          display TAB characters as ^I
    -u                       (ignored)
    -v, --show-nonprinting   use ^ and M- notation, except for LFD and TAB
    --help               display this help and exit
    --version            output version information and exit
```

Index of Annotated Commands Table Of Contents

```
cc
    C compiler, can be used with A LOT of arguments, do a man cc to find
    out just how many, it's normally used to compile a .c source file to an
```

```
    executable binary, like this:
    cc -o program program.c
```

Index of Annotated Commands Table Of Contents

```
cd
    change directory, works as this:
    cd /way/to/directory/I_want_to/be/in/

    No further explanation needed.
```

Index of Annotated Commands Table Of Contents

```
chattr
    This is a very powerful command with which you can change the
    attributes on an ext2 file system.
    This means that you can make a file impossible to remove
    for as long as the attributes are there.

    The attributes that can be added or removed are the following:

    A       Don't update atime.
    S       Synchronous updates.
    a       Append only.
    c       Compressed.
    i       Immutable.
    d       No dump.
    s       Secure deletion.
    u       Undeletable.

    So here is an example:

    chattr +iu /etc/passwd

    This makes it impossible to remove the /etc/passwd file unless
    you first do:

    chattr -iu /etc/passwd

    This can also be good for the logs, especially, with the a attribute.
    To see the attributes, use: lsattr

    For more info do: man chattr
```

Index of Annotated Commands Table Of Contents

```
chmod
    chmod is a very useful command, it changes the rights of any file.
    To understand this command you need to understand how the permission
    line works:

    -rwxr-xr-x    1 alien    users          58 Feb  7 13:19 file1
    -rw-r--r--    1 alien    users        3.1k Feb  3 15:47 file2

    Let's break the -rwxr-xr-x down into 4 sections:
    - rwx r-x r-x
    The first - you can not change, that tells what sort of file it is,
    as if it's a special file, a directory or a normal file.
    The second rwx is the rights of the owner of the file.
    The third r-x is the rights the group of the file has.
    And the fourth r-x tells us what right others/anyone else has.
    The rights can be:

    r       read rights.
    w       write rights.
    x       execute rights.
    s       suid (su id, execute with someone else's uid, usually root)
    t       saves the programs text on the swap device
    X       executes file only if it's in a dir that has execute rights

    Then we need to know in what of those 3 last fields to set those
    rights, they can be set to:

    a       all (changes the 3 fields synchronously)
    u       user
    g       group
    o       others/anyone else

    You can add or remove rights with the following arguments:
```

```
+       add a right
-       remove a right
=       absolute right

So say now that we have a file called file1, that looks like this:

-rwxr-xr-x    1 alien    users          58 Feb  7 13:19 file1

And we wanna take away all execution rights.
Then we can either do:
chmod a-x file1
or
chmod ugo-x file1
And if we wanna make a file executable to everyone in its group,
in this case the group "users", then we do:
chmod g+x file1

The other way to do this, is to use octal numbers to set the
rights in the permission line.
This requires a bit more thinking if your not use to it, but here's
how it works:

First we break up the permission line into 3 sections again (not
counting the leading - or d), and then we put numbers on each
of the 3 fields in each of the 3 sections.

- rwx rwx rwx
  421 421 421

Now to change a line to say: -rwxrx-r-x
You would:
x and r in the last field, that would mean 1+4=5, then the same thing
in the middle field, and last we have r, w and x in the first so then
we count them all, 1+2+4=7.
If we now line up our results of this little mathematic we get: 755
And so to change a permission line to -rwxrx-r-x we do:

chmod 755 <file>

Here's how it looks:

Oct     Bin    Rights

0         0     ---
1         1     --x
2        10     -w-
3        11     -wx
4       100     r--
5       101     r-x
6       110     rw-
7       111     rwx

Then we have the suid stuff for this with octal counting, that you set
before the normal rights, I'll explain that in a bit, first here
is the number codes for the special options as suid.

7*** SUID (user & group and set's file +t)
6*** SUID (user & group)
5*** SUID +t (saves the files text to the swap partition and SUID user)
4*** SUID (user)
3*** SUID (group and set's file +t)
2*** SUID (group)
1*** +t (saves the files text to the swap partition)
0*** nothing

Here's how it looks:

Oct     Bin    Rights
-       --- rwx rwx rwx
0         0 --- --- ---
1         1 --- --- --t
2        10 --- --s ---
3        11 --- --s --t
4       100 --s --- ---
5       101 --s --- --t
6       110 --s --s ---
7       111 --s --s --t

So if you have a file that we can call 'foo.sh' and you wanna make
so that only the user has write permissions to it, the user
and group has read and execute permissions, and all others has no
```

```
    rights at all to it.

    Then we would count: others, 0, group 5, user 7, and then to SUID
    the group we add a 2 in front of what we have, which means:

    chmod 2750 foo.sh

    This will make foo.sh's permission line look like this:

    -rwxr-s---

    To do the exact same with characters, you do:

    chmod u+rwx,go-rwx,g+s foo.sh

    The most common permissions for files is

    Executable:      (755)   -rwxr-xr-x
    Non-Executable:  (644)   -rw-r--r--

    The easiest way of setting these is by either do:

    chmod 755 file
    or
    chmod =rwxrxrx file

    chmod 644 file
    or
    chmod =rwrr file

    For more information, do: man chmod
```

Index of Annotated Commands Table Of Contents

```
chown
    chown changes owner of a file, it can actually also change the group.
    it works like this:

    chown user file

    This would change the owner of the file to user, but note that
    you can not change to owner of a file to a user that is owned
    by someone else, same thing is that you can not change another
    users files so that you own them.
    Basically, you need to be root to for this command in most cases.
    If you wanna change both the user and the group of a file, you do
    like this:

    chown user.group file

    That would change the owner of the file to user and the group
    of the file to group.
    For more info on this do: man chown
```

Index of Annotated Commands Table Of Contents

```
chroot
    runs a command or interactive shell with special root directory.
    It works like this:

    chroot /new/root/directory/ command

    This can be good for some programs or commands, that rather
    would have / as root directory then ~/ etc.
```

Index of Annotated Commands Table Of Contents

```
cmp
    compares 2 files for differences, it can be used with the
    following arguments:

    -l    Print the byte number (decimal) and the differing byte
          values (oc- tal) for each difference.

    -s    Print nothing for differing files; return exit status only.

    It works like this:

    cmp file1 file2
```

```
    Or
    cmp -s file1 file2

    Not a very big, but still useful command.
```

Index of Annotated Commands Table Of Contents

```
cp
    copy, copy's a file from one location to another, may also copy
    one filename to another, used as this:
    cp file /some/other/dir/
    or
    cp file file.old
```

Index of Annotated Commands Table Of Contents

```
crontab
    Crontab has already been explained in this tutorial.
```

Index of Annotated Commands Table Of Contents

```
cut
    cut is a very powerful command, that allows you to cut in texts,
    It works like this: cut [arguments] <file>

    -b, --bytes=LIST
            output only these bytes

    -c, --characters=LIST
            output only these characters

    -d, --delimiter=DELIM
            use DELIM instead of TAB for field delimiter

    -f, --fields=LIST
            output only these fields

    -n
            (ignored)

    -s, --only-delimited
            do not print lines not containing delimiters

    --output-delimiter=STRING
            use STRING as the output delimiter

            the default is to use the input delimiter

    --help
            display the help and exit

    --version
            output version information and exit

    One of the many ways to use it is like this, say that you have a file
    named "hostlist" that contains this:

    beta.linux.com has address 216.200.201.197
    shiftq.linux.com has address 216.200.201.195
    irc.linux.com has address 216.200.201.199
    oreilly.linux.com has address 208.201.239.30
    srom.linux.com has address 204.94.189.33
    admin.linux.com has address 216.200.201.194

    And you ONLY wanna list the IP's from it, then you do this:

    cut -d ' ' -f 4 testfile

    That will output only the IP's, first we set the delimiter to ' '
    which means a space, then we display the 4'th field separated by
    the delimiter, which here is the IP's.

    Or that you have a file (say named column.txt) that contains this:

    something                if we have to
    or someone              cut and paste
    likes to write          the columns.
    in columns, we          So what do
    don't like that          we do about
```

```
    especially not          this?

    To cut out each column is done like this:

    cut -c 1-14 column.txt
    cut -c 23-40 column.txt

    This would fist cut the file lengthwise and display characters
    1-14 and then the same thing again but characters 23-40.

    Now a simple way to get them in a long row instead of columns
    in a file is this:

    cut -c 1-14 column.txt > no-columns.txt
    cut -c 23-40 column.txt >> no-columns.txt
```

Index of Annotated Commands Table Of Contents

```
date
    date alone returns the current date and time in the following format:

    day month  date hr:min:sec timezone year

    But can be executed with the following arguments:

    %%   a literal %
    %a   locale's abbreviated weekday name (Sun..Sat)
    %A   locale's full weekday name, variable length (Sunday..Saturday)
    %b   locale's abbreviated month name (Jan..Dec)
    %B   locale's full month name, variable length (January..December)
    %c   locale's date and time (Sat Nov 04 12:02:33 EST 1989)
    %d   day of month (01..31)
    %D   date (mm/dd/yy)
    %e   day of month, blank padded ( 1..31)
    %h   same as %b
    %H   hour (00..23)
    %I   hour (01..12)
    %j   day of year (001..366)
    %k   hour ( 0..23)
    %l   hour ( 1..12)
    %m   month (01..12)
    %M   minute (00..59)
    %n   a newline
    %p   locale's AM or PM
    %r   time, 12-hour (hh:mm:ss [AP]M)
    %s   seconds since 00:00:00, Jan 1, 1970 (a GNU extension)
    %S   second (00..60)
    %t   a horizontal tab
    %T   time, 24-hour (hh:mm:ss)
    %U   week number of year with Sunday as first day of week (00..53)
    %V   week number of year with Monday as first day of week (01..52)
    %w   day of week (0..6);  0 represents Sunday
    %W   week number of year with Monday as first day of week (00..53)
    %x   locale's date representation (mm/dd/yy)
    %X   locale's time representation (%H:%M:%S)
    %y   last two digits of year (00..99)
    %Y   year (1970...)
    %z   RFC-822 style numeric timezone (-0500) (a nonstandard extension)
    %Z   time zone (e.g., EDT), or nothing if no time zone is determinable

    For example, if you want to the time as hr:min:sec day, you would do:

    date +'%H:%M:%S %a'

    Or if you wanted to display the name of the month only, you would do:

    date +%B
```

Index of Annotated Commands Table Of Contents

```
dc
    dc is an arbitrary precision calculator.
    man dc for more info.
```

Index of Annotated Commands Table Of Contents

```
dd
    disk duplicator, this is a very powerful command, that is
    useful for doing backups as well as creating boot floppy's
```

```
    from images.
    Say now that you have a Slackware standard boot floppy image (bare.i)
    and you want to write it to a floppy, then you do this:

    dd if=bare.i of=/dev/fd0 conv=sync

    If you instead have a RedHat or Mandrake boot image, just replace
    the bare.i in the line with boot.img, under the condition that
    you are standing in a directory that contains that specific image.

    The conv=sync part is just there to make sure that the disks are
    synced.

    dd is a quite big command so I suggest you take a look at the man page.
```

Index of Annotated Commands Table Of Contents

```
declare
    declare will declare a variable and may set attributes to it.
    The attributes declare can set or use with the following flags are:

    -p         show variable with attributes.
    -a         to make a variable(s) an array (if supported)
    -f         to select from among function names only
    -F         to display function names without definitions
    -r         to make variable(s) readonly
    -x         to export variable(s)
    -i         to make variable(s) have the `integer' attribute set

    Using `+' instead of `-' turns off the given attribute(s) instead
    of setting them.
    If declare is used within a function, the variables will be
    local, the same way as if the `local` command had been used.

    The -r option works the same as the `readonly` command.
    And the -r option can not be removed once it's set.

    Here's a short example:

    declare -xr foo=bar

    This would do the same as to do:

    export foo=bar; readonly foo

    For more info on this, do: help declare
```

Index of Annotated Commands Table Of Contents

```
depmod
    depmod loads kernel modules, and is a very powerful command,
    its greatest use is that it can reload all kernel modules
    in a single line:

    depmod -a

    This is especially good if you have recompiled some modules and
    installed them, and you don't wanna reboot the system.
    The command also allows you to load single modules or several
    modules, like this:

    depmod module1.o module2.o ... etc.

    For more info, man depmod
```

Index of Annotated Commands Table Of Contents

```
df
    Reports filesystem disk space usage.
    df can be used with the following arguments:

    -a, --all
    include filesystems having 0 blocks

    --block-size=SIZE use SIZE-byte blocks

    -h, --human-readable
    print sizes in human readable format (e.g., 1K 234M 2G)
```

```
    -H, --si
    likewise, but use powers of 1000 not 1024

    -i, --inodes
    list inode information instead of block usage

    -k, --kilobytes
    like --block-size=1024

    -l, --local
    limit listing to local filesystems

    -m, --megabytes
    like --block-size=1048576

    --no-sync
    do  not  invoke  sync  before  getting  usage  info (default)

    -P, --portability
    use the POSIX output format

    --sync invoke sync before getting usage info

    -t, --type=TYPE
    limit listing to filesystems of type TYPE

    -T, --print-type
    print filesystem type

    -x, --exclude-type=TYPE
    limit listing to filesystems not of type TYPE

    -v     (ignored)

    --help display this help and exit

    --version
    output version information and exit

    My favorite is to use: df -h
```

Index of Annotated Commands Table Of Contents

```
dhcpcd
    dhcpcd is used to obtain an IP if you have dynamic IP on a LAN
    such as a cable modem with dynamic IP.
```

Index of Annotated Commands Table Of Contents

```
dialog
    The dialog command has already been explained in this tutorial.
```

Index of Annotated Commands Table Of Contents

```
diff
    diff is a very large command that finds the difference between
    two files, it's very handy to have to make patches.
    The basic use of diff is as follows:
    diff file1 file2
    for more and full info on this command, do: man diff
```

Index of Annotated Commands Table Of Contents

```
dir
    Same as "ls".
```

Index of Annotated Commands Table Of Contents

```
dmesg
    dmesg can print or control the kernel ring buffer.
    by default it'll show a log of loaded and unloaded modules
    and other kernel events, like initialization if RAM disks etc.
    (this is flushed at each reboot)
    This is useful to make a boot.messages file, by simply doing this:
    dmesg > boot.messages
    If there is any errors at the boot up this command is the first
    you would use to try to determine the error.
```

```
    This is the syntax of dmesg (cut'n'paste of the man page):

    dmesg [ -c ] [ -n level ] [ -s bufsize ]

    The options (-c/-n/-s) means the following:

    -c
            clear the ring buffer contents after printing.

    -s bufsize
            use  a  buffer  of bufsize to query the kernel ring
            buffer.  This is 8196 by default (this matches  the
            default  kernel  syslog  buffer  size in 2.0.33 and
            2.1.103).  If you have set  the  kernel  buffer  to
            larger  than  the  default  then this option can be
            used to view the entire buffer.

    -n level
            set the level at which logging of messages is  done
            to  the  console.   For  example, -n 1 prevents all
            messages, expect panic messages, from appearing  on
            the  console.   All  levels  of  messages are still
            written to /proc/kmsg, so syslogd(8) can  still  be
            used  to  control  exactly  where  kernel  messages
            appear.  When the -n option is used, dmesg will not
            print or clear the kernel ring buffer.

            When both options are used, only the last option on
            the command line will have an effect.


    An example of usage is this:

    dmesg -c -s 16392

    This would print the kernel ring buffer (with a buffer size of 16392)
    And then flush the contents.

    For more info on this command do: man dmesg
```

Index of Annotated Commands Table Of Contents

```
do
    do just does what it says, and is used in among others
    'while' loops, if you have read the whole tutorial this far
    (and have photographic memory) you understand what I'm saying.
```

Index of Annotated Commands Table Of Contents

```
domainname
    See hostname.
```

Index of Annotated Commands Table Of Contents

```
du
    du shows estimated file space usage.
    du is a good command to show how much space a directory takes up.
    I prefer to use it with the -h argument (human readable, see df).
    du has lots of arguments, do `man du` for a full list.
```

Index of Annotated Commands Table Of Contents

```
echo
    echo will redisplay anything you put after it.
    This is perhaps the most used command in bash scripting, and
    very useful in everyday *nix handling as well, I'll get back
    to that in a moment. but first, echo has the following arguments:

    -n     do not output the trailing newline

    -e     enable interpretation of the backslash-escaped characters
           listed below

    -E     disable interpretation of those sequences in STRINGs

    --help display this help and exit (should be alone)
```

```
    --version
           output  version  information  and  exit  (should be alone)

    AND these:

    e       backslash
    c       suppress trailing newline
    a       alert (BELL)
    f       newline and vertical tab
    n       new line
    r       delete recursively (rest of line backwards)
    t       vertical tab
    v       newline and vertical tab (vertical tab?)
    xa      new line
    xb      newline and vertical tab
    xc      newline and vertical tab
    xd      delete rest of line forward
    xe      ascii ... screws up the console (type reset to get it back)

    So to get a bell (beep) you just do:

    echo -e "\a"

    Or to screw up your console, do:

    echo -e "\xe"
```

Index of Annotated Commands Table Of Contents

```
eject
    With eject you can eject removable medias, such as tapes, JAZ, ZIP,
    CD-rom and so on.
    The command is pretty self explanatory, and can be used with the
    following arguments:

    -h       --help
    -v       --verbose
    -d       --default
    -a       --auto
    -c       --changerslot
    -t       --trayclose
    -n       --noop
    -r       --cdrom
    -s       --scsi
    -f       --floppy
    -q       --tape

    The eject command is used as follows: eject [argument] <name>
    The name is the name of th drive, either from /dev, /mnt or
    by its mountpoint name.
```

Index of Annotated Commands Table Of Contents

```
else
    Used in 'if' statements, and does what it says, used like this:
    if [ "arg1" = "arg2" ]; then echo "match" ; else echo "no match" ; fi
```

Index of Annotated Commands Table Of Contents

```
env
    Display the environment settings.
    Can be used with the following arguments:
       -i, --ignore-environment. start with an empty environment
       -u, --unset=NAME. remove variable from the environment
       --help display this help and exit
       --version
```

Index of Annotated Commands Table Of Contents

```
exit
    exit is used to kill the current process.
    It can either be used to logout or to kill a running script from
    within the script, in the later case it can be used with a
    return number as argument, ie. exit 0
```

Index of Annotated Commands Table Of Contents

```
expr
    expr is a counter or command line calculator, it can handle most
    simple integer calculations.
    It can use all the normal ways of counting including boolean
    operators, such as | OR, != NOT IS, and so on.
    It's simply used as this: expr 1 + 1
    One thing to remember, since this is in the command line, if you use
    * (times), you have to use it like this: expr 2 '*' 2
    The ' precise quote makes sure that the star is not treated
    as a wildcard.
```

Index of Annotated Commands Table Of Contents

```
fdisk
    fdisk is the classic disk handler, with fdisk you can edit your
    hard drive(s) in a lot of ways, as adding or removing partitions,
    list the partitions and so on.
    You start fdisk as this: fdisk /dev/<disk to veiw/edit>
    This may be a disk such as /dev/hda /dev/hdb /dev/hdc and so on.
    Note that you can not determine a specific HD partition to
    start from, since fdisk operates on the whole HD.

    When you start fdisk you will have the following commands,
    followed by there explanation:

    a   toggle a bootable flag
    b   edit bsd disklabel
    c   toggle the dos compatibility flag
    d   delete a partition
    l   list known partition types
    m   print this menu
    n   add a new partition
    o   create a new empty DOS partition table
    p   print the partition table
    q   quit without saving changes
    s   create a new empty Sun disklabel
    t   change a partitions system id
    u   change display/entry units
    v   verify the partition table
    w   write table to disk and exit
    x   extra functionality (experts only)

    And in 'x' the extra functionality (experts only) mode.

    b   move beginning of data in a partition
    c   change number of cylinders
    d   print the raw data in the partition table
    e   list extended partitions
    g   create an IRIX partition table
    h   change number of heads
    m   print this menu
    p   print the partition table
    q   quit without saving changes
    r   return to main menu
    s   change number of sectors/track
    v   verify the partition table
    w   write table to disk and exit

    For more info on fdisk, do: man fdisk
```

Index of Annotated Commands Table Of Contents

```
file
    The file command will tell you what type of file a file is.
    file basically works like this:
    file [ -bciknsvzL ] [ -f namefile ] [ -m magicfiles ] file
    The options are as follows:

    -b      Do not prepend filenames to  output  lines  (briefmode).

    -c      Cause  a  checking  printout of the parsed form of
            the magic file.  This is usually used in  conjunction
            with  -m  to  debug  a new magic file before installing it.

    -f namefile
            Read the names of the files to  be  examined  from
            namefile  (one per line) before the argument list.
            Either namefile or at least one filename  argument
            must  be  present; to test the standard input, use
```

```
        ``-'' as a filename argument.

-i      Causes  the  file  command  to  output  mime  type
        strings  rather  than  the  more traditional human
        readable  ones.  Thus  it  may  say  ``text/plain;
        charset=us-ascii''  rather than ``ASCII text''. In
        order for this option to work,  file  changes  the
        way  it  handles  files  recognised by the command
        its self (such as many of the  text  file  types,
        directories  etc), and makes use of an alternative
        ``magic'' file.  (See ``FILES'' section, below).

-k      Don't stop at the first match, keep going.

-m list
        Specify an  alternate  list  of  files  containing
        magic  numbers.   This  can be a single file, or a
        colon-separated list of files.

-n      Force stdout to be flushed  after  check  a  file.
        This  is  only useful if checking a list of files.
        It is  intended to be  used by programs  that want
        filetype output from a pipe.

-v      Print the version of the program and exit.

-z      Try to look inside compressed files.

-L      option  causes  symlinks  to  be  followed, as the
        like-named option in ls(1).  (on systems that support
        symbolic links).

-s      Normally, file only attempts to read and determine
        the type of argument files which  stat(2)  reports
        are   ordinary  files.   This  prevents  problems,
        because reading special files  may  have  peculiar
        consequences.   Specifying  the  -s  option causes
        file to also read argument files which  are  block
        or  character  special  files.  This is useful for
        determining the filesystem types of  the  data  in
        raw  disk  partitions,  which  are  block  special
        files.  This option also causes file to  disregard
        the file size as reported by stat(2) since on some
        systems it reports a zero size for raw disk partitions.

Here's a very simple usage example:
file /bin/sh
file script.sh

For more info do: man file
```

Index of Annotated Commands Table Of Contents

```
find
    find is a very powerful and useful command, it is as good for finding
    a file name as to helping you secure your system against hackers.

    find works basically like this: find <path> [argument] <file>

    You REALLY need to read its manual page, if you wanna know
    about this command, but here are some examples:

    Find all files that are set suid root:
    find / -perm +4000

    Find all regular files named core (this will skip directory's):
    find / -type f -name core

    Find all filenames that contains the word 'conf':
    find / -name *conf*

    Find all directory's that ends with 'bin':
    find / -type d -name *bin

    Find all files named test.sh and execute them:
    find / -name test.sh -exec {} \;

    Find all regular files that contains the word .exe and
    remove them by force without asking:
    find / -type f -name *.exe -exec rm {} -rf \;
```

```
    Even if you are root you may come across errors like this:

    find: /proc/10502/fd: Permission denied

    The easiest way to deal with this is to add a: 2>/dev/null
    after your command string, that will direct all such errors
    to /dev/null (the black hole of UNIX :P)
```

Index of Annotated Commands Table Of Contents

```
ftpwho
    ftpwho is a command where you can see how many users there are
    logged on to your ftp, under the condition that you have an
    ftp server on your system that is.
```

Index of Annotated Commands Table Of Contents

```
g++
    GNU C++ compiler.
    See its man page.
```

Index of Annotated Commands Table Of Contents

```
gcc
    GNU C Compiler.
    See cc
    And see the gcc man page
```

Index of Annotated Commands Table Of Contents

```
gdb
    GNU Debugger, has A LOT of commands and arguments,
    see: man gdb.
```

Index of Annotated Commands Table Of Contents

```
gpm
    gpm is the Linux mouse daemon, it's unspareble when it comes to
    working an a console, cut & paste is a wonderful thing.
    gpm works basically as this: gpm [options]
    The most common options would be as this:

    gpm -m /dev/mouse -t ps2

    This would start a PS/2 mouse, under the conditions that the
    PS/2 port (/dev/psaux) is linked to the mouse device (/dev/mouse).

    You can use "gpm -m /dev/psaux -t ps2" just as well.
    Or if you have a serial mouse no COM1 you can start it like this:

    gpm -m /dev/cua0 -t ms

    The -m argument means, the mouse device, and the -t argument is
    the protocol it's going to use.

    For a list of all gpm's arguments do: gpm -h
    And for a list of all the possible mouse protocols, do: gpm -t help

    The basic console cut & paste functions for a 3 button mouser is:

    Left button - hold and drag to highlight text (copy's text to memory).

    Middle button - pastes text that are in memory (see left button)

    Right button - mark a starting point with a single left click and then
    mark an end point with the right button to highlight the whole section.

    Once you get it to work, you may add the line to: /etc/rc.d/rc.local
```

Index of Annotated Commands Table Of Contents

```
grep
    grep is another of the very powerful commands
```

Index of Annotated Commands Table Of Contents

```
halt
    This will halt (shutdown -h now) your system.
```

Index of Annotated Commands Table Of Contents

```
hdparm
    hdparm is a powerful tool to control your hard drives.
    It works like this: hdparm <arguments> <hard drive>
    The arguments can be:

    -a   get/set fs readahead
    -A   set drive read-lookahead flag (0/1)
    -c   get/set IDE 32-bit IO setting
    -C   check IDE power mode status
    -d   get/set using_dma flag
    -D   enable/disable drive defect-mgmt
    -E   set cd-rom drive speed
    -f   flush buffer cache for device on exit
    -g   display drive geometry
    -h   display terse usage information
    -i   display drive identification
    -I   read drive identification directly from drive
    -k   get/set keep_settings_over_reset flag (0/1)
    -K   set drive keep_features_over_reset flag (0/1)
    -L   set drive doorlock (0/1) (removable harddisks only)
    -m   get/set multiple sector count
    -n   get/set ignore-write-errors flag (0/1)
    -p   set PIO mode on IDE interface chipset (0,1,2,3,4,...)
    -P   set drive prefetch count
    -q   change next setting quietly
    -r   get/set readonly flag (DANGEROUS to set)
    -R   register an IDE interface (DANGEROUS)
    -S   set standby (spindown) timeout
    -t   perform device read timings
    -T   perform cache read timings
    -u   get/set unmaskirq flag (0/1)
    -U   un-register an IDE interface (DANGEROUS)
    -v   default; same as -acdgkmnru (-gr for SCSI, -adgr for XT)
    -V   display program version and exit immediately
    -W   set drive write-caching flag (0/1) (DANGEROUS)
    -X   set IDE xfer mode (DANGEROUS)
    -y   put IDE drive in standby mode
    -Y   put IDE drive to sleep
    -Z   disable Seagate auto-powersaving mode

    Some examples:
    hdparm -Tt /dev/hda      (Time the cache/device read times)
    hdparm -c 1 /dev/hda     (This made my HD read the cache twice as fast)
    hdparm -Yy /dev/hda      (This will totally power down the HD until
                             it's needed, very useful to save power
                             or if you just need a minutes silence)

    For more info: man hdparm
```

Index of Annotated Commands Table Of Contents

```
head
    the head command by default brings up the 10 top lines of a file,
    but can be used with these arguments:

    -<n>                      where the <n> is the number of lines to get
    -c, --bytes=SIZE          print first SIZE bytes
    -n, --lines=NUMBER        print first NUMBER lines instead of first 10
    -q, --quiet, --silent     never print headers giving file names
    -v, --verbose             always print headers giving file names
        --help                display this help and exit
        --version             output version information and exit

    Here's some examples:

    head file

    head -1 file

    head -50 file

    head -c 100 file

    This command can prove to be very useful.
```

Index of Annotated Commands Table Of Contents

```
help
    help is a command that shows information on built in commands.
    like ., cd, jobs, %, test, etc.
    It works like this: help <command>
```

Index of Annotated Commands Table Of Contents

```
hexdump
    hexdump is a command that will give a hex dump of any file.
    For more info on this command do: man hexdump
```

Index of Annotated Commands Table Of Contents

```
hexedit
    hexedit is a hex editor, very good for debugging binaries,
    hexedit has a lot of internal commands, do: man hexedit
    for more help on it.
```

Index of Annotated Commands Table Of Contents

```
hostname
    With no arguments it displays the current hostname.
    But can also set a new hostname, here are its arguments:

    -s, --short           short host name
    -a, --alias           alias names
    -i, --ip-address      addresses for the hostname
    -f, --fqdn, --long    long host name (FQDN)
    -d, --domain          DNS domain name
    -y, --yp, --nis       NIS/YP domainname
    -F, --file            read hostname or NIS domainname from given file

    Here's an example if you wanna change your hostname:

    hostname -F /etc/HOSTNAME
```

Index of Annotated Commands Table Of Contents

```
id
    Shows you a users ID, default your user UID, GID and group name.
    The command has some arguments it can be used with, like this:

    id [argument] <user>

    Here are the arguments:

    -a               ignore, for compatibility with other versions
    -g, --group      print only the group ID
    -G, --groups     print only the supplementary groups
    -n, --name       print a name instead of a number, for -ugG
    -r, --real       print the real ID instead of effective ID, for -ugG
    -u, --user       print only the user ID
    --help       display this help and exit
    --version    output version information and exit

    So `id -u` will return '0' if you are root (same as `echo $UID`).
```

Index of Annotated Commands Table Of Contents

```
ifdown
    ifdown is a command that will let you shutdown (deactivate) any
    ethernet device. I works as this: ifdown <device>
    So say that you have an eth0 running that you wanna shut down, then
    you just do this: ifdown eth0

    For more info on how to set up an ethernet device,
    see section 7 (Networking) in this tutorial.
```

Index of Annotated Commands Table Of Contents

```
ifup
    ifup works the same as ifdown, but activates the ethernet device
    rather than deactivate it.
```

```
    For more info on how to set up an ethernet device,
    see section 7 (Networking) in this tutorial.
```

Index of Annotated Commands Table Of Contents

```
init
    init sets the runlevel for you.
    If you have read the whole of this tutorial to this point you know
    about where to look for what they mean.
    So if you do: init 0
    The system will shutdown and halt there.
    And if you type: init 6
    The system will reboot, etc.
```

Index of Annotated Commands Table Of Contents

```
insmod
    insmod tries to install a loadable module in the running kernel.
    It works like this:

    insmod [arguments] <-o module_name> object_file [ sym-bol=value ... ]

    Here are the possible arguments:
    -f, --force         Force loading under wrong kernel version
    -k, --autoclean     Make module autoclean-able
    -m                  Generate load map (so crashes can be traced)
    -o NAME
    --name=NAME     Set internal module name to NAME
    -p, --poll          Poll mode; check if the module matches the kernel
    -s, --syslog        Report errors via syslog
    -v, --verbose       Verbose output
    -V, --version       Show version
    -x                  Do not export externs
    -X                  Do export externs (default)

    An example of how to use this is:

    insmod -o 3c90x /lib/modules/2.2.14/net/3c90x.o

    This would load the 3c90x.o module with 3c90x as name.
```

Index of Annotated Commands Table Of Contents

```
install
    install is a command that installs a file properly,
    it works like this: install [arguments] source destination
    The arguments can be any of the following:

    -b, --backup
            make backup before removal

    -c
            (ignored)

    -d, --directory
            treat all arguments as directory names; create all
                    components of the specified directories

    -D
            create all leading components of DEST except the last,
            then copy SOURCE to DEST;  useful in the 1st format

    -g, --group=GROUP
            set group ownership, instead of process' current group

    -m, --mode=MODE
            set permission mode (as in chmod), instead of rwxr-xr-x

    -o, --owner=OWNER
            set ownership (super-user only)

    -p, --preserve-timestamps
            apply access/modification times of SOURCE files to
            corresponding destination files

    -s, --strip
            strip symbol tables, only for 1st and 2nd formats
```

```
    -S, --suffix=SUFFIX
            override the usual backup suffix

    --verbose
            print the name of each directory as it is created

    -V, --version-control=WORD
            override the usual version control

    --help
            display the help and exit

    So if we have a file called foo and we want to install it in
    /usr/local/bin/, and we want it to have the following permission line:
    -rwxr-x---, then we want it to belong to the group ftp, then
    we do like this:

    install -m 750 foo -g ftp /usr/local/bin/

    We could also use:

    install -m u+rwx,g+rx foo -g ftp /usr/local/bin/

    Which would produce the same permission line.

    The install command is good to use if you ever do anything that
    needs to be installed to the system, in a proper way.
```

Index of Annotated Commands Table Of Contents

```
ipchains
    ipchains is a firewall/wrapper that has A LOT of argument,
    it's one of those huge commands, do: man ipchains
    for more information on this command.
```

Index of Annotated Commands Table Of Contents

```
ispell
    Interactive Spell check, this is a useful little command,
    its basic usage is: ispell <file>
    It has the following commands:

    R       Replace the misspelled word completely.
    Space   Accept the word this time only.
    A       Accept the word for the rest of this session.
    I       Accept the word, and put it in your private dictionary.
    U       Accept and add lowercase version to private dictionary.
    0-n     Replace with one of the suggested words.
    L       Look up words in system dictionary.
    X       Write the rest of this file, ignoring misspellings,
            and start next file.
    Q       Quit immediately.  Asks for confirmation.
            Leaves file unchanged.
    !       Shell escape.
    ^L      Redraw screen.
    ^Z      Suspend program.
    ?       Show the help screen.

    Just run it on a file and test it for your self.
```

Index of Annotated Commands Table Of Contents

```
kill
    kill is a very powerful command that can (if you're root)
    kill any running process no the system.
    it works as: kill -<signal> <PID>

    Pid is short for Process ID, which you get with the `ps` command.

    The signals can be any of the following:

    POSIX signals:

    Signal     Value     Action    Comment
    ----------------------------------------------------------------------
    SIGHUP        1        A     Hangup detected on controlling terminal
                                 or death of controlling process
    SIGINT        2        A     Interrupt from keyboard
    SIGQUIT       3        C     Quit from keyboard
```

```
    SIGILL        4        C     Illegal Instruction
    SIGABRT       6        C     Abort signal from abort(3)
    SIGFPE        8        C     Floating point exception
    SIGKILL       9       AEF    Kill signal
    SIGSEGV      11        C     Invalid memory reference
    SIGPIPE      13        A     Broken pipe: write to pipe with no readers
    SIGALRM      14        A     Timer signal from alarm(2)
    SIGTERM      15        A     Termination signal
    SIGUSR1   30,10,16     A     User-defined signal 1
    SIGUSR2   31,12,17     A     User-defined signal 2
    SIGCHLD   20,17,18     B     Child stopped or terminated
    SIGCONT   19,18,25           Continue if stopped
    SIGSTOP   17,19,23    DEF    Stop process
    SIGTSTP   18,20,24     D     Stop typed at tty
    SIGTTIN   21,21,26     D     tty input for background process
    SIGTTOU   22,22,27     D     tty output for background process

    Non-POSIX signals:

    Signal       Value     Action   Comment
    ----------------------------------------------------------------------
    SIGBUS      10,7,10      C     Bus error (bad memory access)
    SIGPOLL                  A     Pollable event (Sys V). Synonym of SIGIO
    SIGPROF     27,27,29     A     Profiling timer expired
    SIGSYS      12,-,12      C     Bad argument to routine (SVID)
    SIGTRAP        5         C     Trace/breakpoint trap
    SIGURG      16,23,21     B     Urgent condition on socket (4.2 BSD)
    SIGVTALRM   26,26,28     A     Virtual alarm clock (4.2 BSD)
    SIGXCPU     24,24,30     C     CPU time limit exceeded (4.2 BSD)
    SIGXFSZ     25,25,31     C     File size limit exceeded (4.2 BSD)

    Other signals:

    Signal       Value     Action   Comment
    -------------------------------------------------------------------
    SIGIOT         6         C     IOT trap. A synonym for SIGABRT
    SIGEMT       7,-,7
    SIGSTKFLT    -,16,-      A     Stack fault on coprocessor
    SIGIO       23,29,22     A     I/O now possible (4.2 BSD)
    SIGCLD       -,-,18            A synonym for SIGCHLD
    SIGPWR      29,30,19     A     Power failure (System V)
    SIGINFO      29,-,-            A synonym for SIGPWR
    SIGLOST      -,-,-       A     File lock lost
    SIGWINCH    28,28,20     B     Window resize signal (4.3 BSD, Sun)
    SIGUNUSED    -,31,-      A     Unused signal (will be SIGSYS)


    When you use the kill you can either use the numeric code, as
    say that we have a PID 1234 that we wanna kill, then we can either
    do: kill -9 1234 or we can do: kill -KILL 1234
    So you don't have to include that leading SIG in the signals
    when you use them by name.
```

Index of Annotated Commands Table Of Contents

```
killall
    killall is the same as kill but kills processes by name,
    As say that you have 10 processes running all named: httpd
    and you wanna kill them all in one command.
    Then: killall -9 httpd  would be the way to go about it.
```

Index of Annotated Commands Table Of Contents

```
lastlog
    lastlog is a command that shows you a list of the users and when
    they last logged in, from what host and on which port.
    lastlog can be used with the following arguments:

    -u      username
    -t      number of days

    so if I wanna check if a user named 'user' has logged in during
    the last 50 days I do: lastlog -u user -t 50
```

Index of Annotated Commands Table Of Contents

```
ldconfig
    ldconfig updates the list of directory's in where library's can be
    found as /lib and /usr/lib, if you wanna add a directory to this you
```

```
    can add them in /etc/ld.so.conf
    By just typing `ldconfig` you will update this, but it can
    also be executed with more arguments, for more info on this
    command do: man ldconfig

    Just note that this is not really a command that you will use
    every day.
```

Index of Annotated Commands Table Of Contents

```
ldd
    ldd can check what libraries a dynamically executable file needs.
    and it can have the following switches:

    --help                  print this help and exit
    --version               print version information and exit
    -d, --data-relocs       process data relocations
    -r, --function-relocs   process data and function relocations
    -v, --verbose           print all information

    It works like this:

    ldd <file>

    Example:

    ldd /sbin/ifconfig
```

Index of Annotated Commands Table Of Contents

```
less
    less is more than more ..... ummmm
    less works a bit like cat but it will stop at each screen and you
    can scroll up and down in the file to view its contents,
    it works basically like this: less <textfile>
    Do a: less --help
    For a full index of its commands, and note that you get out
    of less by pressing the letter 'q'.
```

Index of Annotated Commands Table Of Contents

```
lilo
    lilo is the LInux LOader, and is on most distros the default
    boot loader, with lilo you can rewrite your boot sector and
    everything that involves your booting or switching between
    several installed operating systems, lilo's configuration file
    is /etc/lilo.conf
    for more info about lilo and what lilo can do, do: man lilo
```

Index of Annotated Commands Table Of Contents

```
ln
    link, with ln you can link any file, this is essential to *nix
    as, say that you have a config file that needs to be in the same
    dir as its program but you want it in /etc with all the other
    configuration files, then you can link it to /etc so the link
    appears in /etc and works just like the real file.

    Usually ln is used to set symbolic links (sym links) where you
    can see the difference of the link and the file and you can remove
    the link without it affecting the real file.
    A symbolic link is set in this way: ln -s file link
```

Index of Annotated Commands Table Of Contents

```
lndir
    link directory, about the same as ln but links directory's,
    see the: man lndir
```

Index of Annotated Commands Table Of Contents

```
loadkeys
    loadkeys, basically works like: loadkeys /usr/lib/kbd/keymaps/<keymap>
    but also has some arguments (that I never used), if you want
    more info: man loadkeys
```

Index of Annotated Commands Table Of Contents

```
locate
    locate can locate any file that is read into a database,
    you update this database if you as root type: updatedb
    locate works basically like: locate <whatever-you-wanna-find>
    but can be executed with a lot of arguments, do: locate --help
    or for more info: man locate
```

Index of Annotated Commands Table Of Contents

```
logout
    logout does what it says, it logs you off the shell.
```

Index of Annotated Commands Table Of Contents

```
lpq
    line printer que, checks if you have any printer jobs on que.
```

Index of Annotated Commands Table Of Contents

```
lpr
    line printer, has a lot of commands, but basically works as: lpr <file>
    to print a file, the lpr command has a lot of arguments,
    do: man lpr for more info.
```

Index of Annotated Commands Table Of Contents

```
lprm
    line printer remove, removes any qued jobs (lpq) by there entry number.
```

Index of Annotated Commands Table Of Contents

```
ls
    this is the most basic of all basic commands to know.
    ls lists the contents of a directory, if you type just `ls`
    it will list the contents of the current directory, but it can also
    be used as `ls /way/to/some/other/dir/` to list the contents of
    some other directory, ls has a lot of arguments which are:

    -a, --all
            do not hide entries starting with .

    -A, --almost-all
            do not list implied . and ..

    -b, --escape
            print octal escapes for nongraphic characters

    --block-size=SIZE
            use SIZE-byte blocks

    -B, --ignore-backups
            do not list implied entries ending with ~

    -c
            sort by change time; with -l: show ctime

    -C
            list entries by columns

    --color[=WHEN]
            control whether color is used to distinguish file
            types.  WHEN may be `never', `always', or `auto'

    -d, --directory
            list directory entries instead of contents

    -D, --dired
            generate output designed for Emacs' dired mode

    -f
            do not sort, enable -aU, disable -lst

    -F, --classify
            append indicator (one of */=@|) to entries
```

```
    --format=WORD
        across -x, commas -m, horizontal -x, long -l,
        single-column -1, verbose -l, vertical -C

    --full-time
        list both full date and full time

-g
        (ignored)

-G, --no-group
        inhibit display of group information

-h, --human-readable
        print sizes in human readable format (e.g., 1K 234M 2G)

-H, --si
        likewise, but use powers of 1000 not 1024

    --indicator-style=WORD
        append indicator with style WORD to entry names:
        none (default), classify (-F), file-type (-p)

-i, --inode
        print index number of each file

-I, --ignore=PATTERN
        do not list implied entries matching shell PATTERN

-k, --kilobytes
        like --block-size=1024

-l
        use a long listing format

-L, --dereference
        list entries pointed to by symbolic links

-m
        fill width with a comma separated list of entries

-n, --numeric-uid-gid
        list numeric UIDs and GIDs instead of names

-N, --literal
        print raw entry names (don't treat e.g. control
        characters specially)

-o
        use long listing format without group info

-p, --file-type
        append indicator (one of /=@|) to entries

-q, --hide-control-chars
        print ? instead of non graphic characters

    --show-control-chars
        show non graphic characters as-is (default)

-Q, --quote-name
        enclose entry names in double quotes

    --quoting-style=WORD
        use quoting style WORD for entry names:
        literal, shell, shell-always, c, escape

-r, --reverse
        reverse order while sorting

-R, --recursive
        list subdirectories recursively

-s, --size
        print size of each file, in blocks

-S
        sort by file size

   --sort=WORD
        extension -X, none -U, size -S, time -t, version -v
```

```
            status -c, time -t, atime -u, access -u, use -u

        --time=WORD
            show time as WORD instead of modification time:
            atime, access, use, ctime or status; use
            specified time as sort key if --sort=time

    -t
            sort by modification time

    -T, --tabsize=COLS
            assume tab stops at each COLS instead of 8

    -u
            sort by last access time; with -l: show atime

    -U
            do not sort; list entries in directory order

    -v
            sort by version

    -w, --width=COLS
            assume screen width instead of current value

    -x
            list entries by lines instead of by columns

    -X
            sort alphabetically by entry extension

    -1
            list one file per line

        --help                display this help and exit
        --version             output version information and exit

    Some good examples are:

    ls -la
    ls -laF
    ls -laF --color
    ls -d */

    Also see earlier in this tutorial about the `alias` command
```

Index of Annotated Commands Table Of Contents

```
lsattr
    list attributes, this command lists a files file system attributes.
    For more info see: man lsattr
```

Index of Annotated Commands Table Of Contents

```
lsmod
    list modules, lists all loaded modules with a very brief information.
```

Index of Annotated Commands Table Of Contents

```
lsof
    list open files, this is a huge command, so if you really
    wanna find out more about this interesting command you will have
    to read the manual page for it.
    But here's an example of use for it:

    lsof -p 1

    Which would be the same as:

    lsof -p `pidof init`

    Here's another example:

    lsof -p `pidof httpd | sed 's/\ /,/g'`

    The "-p" means that the following argument will be a PID (Process ID).
    The "sed" part in the later example replaces any spaces with "," since
    lsof doesn't want spaces between the pids, as the output of pidof gives.
```

```
    For more info see: man lsof
```

Index of Annotated Commands Table Of Contents

```
lynx
    lynx is a console based world wide web browser, that has a lot of
    arguments with which it can be executed, but it basically works like
    this: lynx <url>
    If you press 'g' while in lynx you can type in the url where you
    wanna go, and if you press 'q' you quit lynx.
    You search in text with lynx with '/' and move around with
    the arrow keys and the TAB key.

    A tips is that lynx works as a file manager, as this: lynx </path/>

    A good usage for lynx is that you can use it as direct downloader,
    like this: lynx -source ftp://ftp.bogus.com/foo/bar.tar.gz > bar.tar.gz

    For more help or information do: lynx --help
    Or: man lynx
```

Index of Annotated Commands Table Of Contents

```
mail
    mail is most commonly used to just check your mail in the most
    simple way by just typing `mail`, but it can also be used
    with a lot of arguments, I have personally never used
    any arguments to the mail command, but if you wanna check
    them out do: man mail
```

Index of Annotated Commands Table Of Contents

```
man
    manual pages, there are several different manual pages, say for example
    the command exec, `man exec` should bring you little, while
    `man 3 exec` should bring you the C function manual on exec.

    The man pages traditional way of storing is:

    man1    misc user commands
    man2    C programming functions
    man3    more C programming functions
    man4    network related manuals
    man5    system related files
    man6    game manuals
    man7    misc technical manuals
    man8    misc superuser commands
    man9    misc system/devices

    I may be wrong about the category's there, but that is how it seems
    to me.
    Anyway, to bring up a manual page simply do: man <command>
    or: man <number> <command>
```

Index of Annotated Commands Table Of Contents

```
mc
    midnight commander is a visual shell for *nix Operating Systems.
    mc is quite large and has a lot of arguments, I personally don't use
    midnight commander at all, but if you wanna learn more about it
    do: man mc
```

Index of Annotated Commands Table Of Contents

```
mesg
    mesg is a command with which you control if other users should
    have write access to your terminal, as `wall` messages, `write`
    or anything similar.

    mesg y  turns on the access for others to write to your terminal.
    mesg n  turns off the access for others to write to your terminal.
```

Index of Annotated Commands Table Of Contents

```
mkdir
```

```
    make directory, creates a directory, works as: mkdir [arguments] dir/
    The arguments can be as follows:

    -m, --mode=MODE         see chmod's octal (numerical) modes

    -p, --parents           no error if existing, make  parent
                            directories  as needed

    --verbose               print a message for each created directory

    --help                  display the help and exit

    --version               output version information and exit

    mkdir is most commonly used as: mkdir <newdir>
```

Index of Annotated Commands Table Of Contents

```
mknod
    mknod is used to create special files, as devices.
    mknod's syntax is this: mknod [arguments] <name> <type> [MAJOR MINOR]
    It can be used with the following arguments:

    -m, --mode=MODE
            set permission mode (as in chmod), not 0666 - umask

    --help
            display this help and exit

    --version
            output version information and exit

    MAJOR MINOR are forbidden for <type> p, else they must be used.

    b      create a block (buffered) special file

    c, u   create a character (unbuffered) special file

    p      create a FIFO

    You need to know the devices major/minor number if you gonna use this
    command, those are located in /usr/src/linux/Documentation/devices.txt
    that comes with the kernel source.
    The "char" is the minor and the number before the devices are the
    major numbers so say that you wanna make a new /dev/null for some
    reason, then you read the devices.txt and see this:


    1 char        Memory devices
                    1 = /dev/mem          Physical memory access
                    2 = /dev/kmem         Kernel virtual memory access
                    3 = /dev/null         Null device

    And so you make the null device like this:

    mknod /dev/null b 1 3

    Or if you wanna make a new /dev/scd device to support another
    emulated scsi cdrom device. (there are 7 scd devices default)
    So here's how you make another:

    mknod /dev/scd8 b 11 8

    This is not as hard at all .....

    for more info: info mknod
    or: man mknod
```

Index of Annotated Commands Table Of Contents

```
modprobe
    modprobe loads modules in a similar way as depmod.
    See modprobe's manual page: man modprobe
```

Index of Annotated Commands Table Of Contents

```
more
    more is a command to display a files contents, it's very similar
    to the `less` command.
```

```
    See `less` and more's manual pages: man more
```

Index of Annotated Commands Table Of Contents

```
mount
    mount, mounts a media, that is to say that you make the contents
    of say a hard drive visible to the system on some mountpoint,
    ie. mount -t vfat /dev/hda1 /windows
    This command would mount hda1 (the first harddrive's (hd a)
    first partition (hda 1), as (-t <filesystem>) vfat which is
    the windows native filesystem.
    Linux native filesystem is ext2.

    mount has A LOT of arguments, if you wanna read about them all
    do: man mount
```

Index of Annotated Commands Table Of Contents

```
mv
    mv, moves a file or directory.
    It works like this: mv [argument] <file-to-move> <new-name/location>
    This is an example: mv /home/alien/bash.tutor /home/old/bash.tutor
    Or just to rename a file: mv bash.tutor bash.file

    mv can also be executed with a lot of arguments, which are:

    -b, --backup
            make backup before removal

    -f, --force
            remove existing destinations, never prompt

    -i, --interactive
            prompt before overwrite

    -S, --suffix=SUFFIX
            override the usual backup suffix

    -u, --update
            move only older or brand new non-directories

    -v, --verbose
            explain what is being done

    -V, --version-control=WORD
            override the usual version control

    --help
            display the help and exit

    --version
            output version information and exit


    Here's an example: mv -f /home/alien/bash.tutor /
    This will by force mv the file to / (if you have write rights to /)
```

Index of Annotated Commands Table Of Contents

```
nc / netcat
    netcat is by default usually located in: /usr/lib/linuxconf/lib/
    netcat is very useful in internet based shell scripts, since
    it can listen on a socket or send to sockets, depending on the version.
    the default netcat can as far as I know only send to sockets.
    works basically like this:

    /usr/lib/linuxconf/lib/netcat --file <file> <ip> <port>

    But can be executed with the following arguments:
    --head <nb_lines>
    --tail <nb_lines>
    --send <file>

    A tip is to make one or two links from /usr/lib/linuxconf/lib/netcat
    to /usr/local/bin/netcat and perhaps /usr/local/bin/nc
```

Index of Annotated Commands Table Of Contents

```
ncftp
    ncftp is a very powerful ftp client.
    ncftp has the following syntax: ncftp [arguments] <host>
    If no arguments is given it will try to login as anonymous user
    with an e-mail as password.

    Most common non-anonymous usage is this: ncftp -u <username> <host>

    The commands you will use the most once logged on to an ftp
    is the following:

    get <name>      download a file
    put <name>      upload a file
    ls              list current directory
    cd  <dir-name>  change directory
    lls             list local directory
    lcd             change local directory

    If you want to read all ncftp's commands and arguments do: man ncftp
```

Index of Annotated Commands Table Of Contents

```
ncftpget
    ncftpget is a command line based ftp download client. It works
    like this: ncftpget [arguments] <host> <local-dir> <remote-files>

    ncftpget comes with ncftp, if you want to see all its commands,
    do: man ncftpget
```

Index of Annotated Commands Table Of Contents

```
ncftpput
    ncftpput is a command line based ftp upload client. It works
    like this: ncftpput [arguments] <host> <remote-dir> <remote-files>

    ncftpput comes with ncftp, if you want to see all its commands,
    do: man ncftpput
```

Index of Annotated Commands Table Of Contents

```
netstat
    netstat will show you the network connections to and from your computer
    that is currently active, it can simply be used by typing `netcat`
    or it can me used with its arguments, if you wanna learn
    more about this command, do: man netcat
```

Index of Annotated Commands Table Of Contents

```
nice
    nice is a command that can set the priority (cpu time) of a program
    or a command, the priorities can be from -20 which is max priority
    to 19 which is the minimum priority.
    nice works like this: nice [argument] <command> <argument>
    The arguments "[argument]" for nice can be:

    -ADJUST
            increment priority by ADJUST first

    -n, --adjustment=ADJUST
            same as -ADJUST

    --help
            display the help and exit

    --version
            output version information and exit

    Example: nice -n -20 make bzImage
    This will make the kernel with as much CPU as it can.
    This means this process has more rights than any other process.

    Another example is: nice -n 19 zgv
    This will give zgv absolutely lowest priority, and will there for
    be the slowest moving processes on the system, as if it runs
    with nice 19 and another process comes and wants more CPU power
    then there is free, `zgv` will in this case give the other
    process of its own power.
```

Index of Annotated Commands Table Of Contents

```
nmap
    nmap is getting to come as default for some Linux distributions,
    and is a port scanner, maybe the best port scanner there is.

    nmap is used like this: nmap [arguments] <host / ip>
    So say you want to port scan yourself you could do:
    nmap 127.0.0.1
    Or: nmap localhost
    The most commonly used arguments to nmap is the '-sS' which
    is a SYN scan, and will in most cases not reveal your IP
    to the one that your scanning, BUT if the other side
    has any kind of modern logging device as a fairly new
    firewall or port logger your IP will be shown to him anyway.

    The other perhaps next most common argument to use is the '-O'
    argument, which will give you a good guess of what the remote
    operating system is this function works the same as for
    the operating system guess program `queso`.

    Example: nmap -sS -O localhost > localhost.log

    The '> localhost.log' part will put the outcome of the scan in a file
    called localhost.log.
```

Index of Annotated Commands Table Of Contents

```
ntpdate
    ntpdate has no manual page nor any help page what I can found,
    perhaps I'll write one if I'm bored some day .....

    ntpdate will synchronize your computers system clock with an atomic
    clock.

    ntpdate's help usage gives this:

    usage: ntpdate [-bBdqsv] [-a key#] [-e delay] [-k file] [-p samples]
    [-o version#] [-r rate] [-t timeo] server ...

    I only use it as: ntpdate <server>
    Like this: ntpdate ntp.lth.se
```

Index of Annotated Commands Table Of Contents

```
ntsys / ntsysv
    runlevel configuration tool.
    This tool lets you configure what services that should be
    started with your runlevel, at least ntsysv has a nice
    ncurses interface that is easy to handle.

    For more information on this command do: man ntsys
    Or: man ntsysv
    Depending on your system.
```

Index of Annotated Commands Table Of Contents

```
objdump
    objdump is a quite large command, that allows you to dump objects
    out of a binary file.
    To dump all objects do: objdump --source <binary file>
    For more info do: man objdump
```

Index of Annotated Commands Table Of Contents

```
passwd
    passwd is a little tool to set a password to a user account,
    it basically works like this: passwd [arguments] <username>
    or if you just type `passwd` you will change your own password.
    passwd can be sued with the following arguments:

    -d, --delete        delete the password for the named account
                        (root only)
    -f, --force         force operation
    -k, --keep-tokens   keep non-expired authentication tokens
    -l, --lock          lock the named account (root only)
    -S, --status        report password status on the named account
```

```
                    (root only)
        --stdin          read new tokens from stdin (root only)
    -u, --unlock         unlock the named account (root only)

    Help options

    -?, --help           Show the help message
    --usage              Display brief usage message

    You still need to do a: man passwd
```

Index of Annotated Commands Table Of Contents

```
patch
    patch simply works like this: patch <original-file> <patch-file>
    A patch is done with the `diff` command as
    this: diff file1 file2 > patchfile
    So then to make file1 identical to file2: patch file1 patchfile

    patch can however be used with a whole lot of arguments,
    if you are interested do: man patch
    Or: patch --help
```

Index of Annotated Commands Table Of Contents

```
pidof
    pidof simply gives the PID of a running process without you having
    to use "ps", say that you want to find out what pid your init has,
    (it will always be one for the init), then you do: pidof init
    Or if you wanna find out which pids are used by the web server (httpd)
    then you do: pidof httpd
    So basically you find out the pids from the process name(s).

    pidof has the following switches:

    -s     Single shot - this instructs the program to only return one pid.

    -x     Scripts too - this causes the program to also return process
           id's of shells running the named scripts.

    -o     Tells pidof to omit processes with that process id.
           The special pid %PPID can be used to name the parent process of
           the pidof program, in other words the calling shell or shell
           script.

    For more info see: man pidof
```

Index of Annotated Commands Table Of Contents

```
ping
    ping is a pretty basic command, that will work
    as: ping [arguments] <ip-or-host>
    The arguments can be as follows:

    -c      <number> count pings to send
    -d      debug
    -f      ping flood
    -i      <number> wait number of seconds between each ping
    -l      <number> preload number of pings
    -n      numeric IP's only
    -p      pattern (in hex) to send as pad code in the ping header
    -q      quiet
    -R      record route
    -s      <number> packet size in bytes
    -v      Verbose output

    So say that you wanna send 5 pings that is 128 bytes each to
    IP 127.0.0.1, then you would do: ping -s 128 -c 5 127.0.0.1
```

Index of Annotated Commands Table Of Contents

```
pmake
    pmake is *BSD make (so I'm told), see make and: man pmake
```

Index of Annotated Commands Table Of Contents

```
pnpdump
```

```
    pnpdump gives a dump of all ISA pnp devices, good to use
    with isapnp etc.
    This is the command you wanna have a look at if your either
    looking for exact info of some ISA device that is pnp, or
    if your system has problems finding a ISA pnp device.
    See the manual pages.
```

Index of Annotated Commands Table Of Contents

```
portmap
    portmap is the server that maps all RPC services, so if you
    wanna use any RPC service you wanna have portmap running.
    For more info: man portmap
```

Index of Annotated Commands Table Of Contents

```
ps
    ps gives you the process list, as in showing you the running
    processes with there pid and other info.
    do: ps --help
    or: man ps      for more info on what arguments it can be executed
    with, personally I use: `ps aux` and `ps x` the most.
```

Index of Annotated Commands Table Of Contents

```
pstree
    process tree, a bit more (ascii) graphical version of ps,
    do: pstree --help
    or: man pstree
    for more help on the arguments, personally I use it alone without
    arguments.
```

Index of Annotated Commands Table Of Contents

```
pwd
    print working directory, shows you your current directory.
    This command can be useful for 2 things what I know of,
    one is to show you where you are, and the other in scripts
    to do say: echo "output will go to: `pwd`/logfile"
```

Index of Annotated Commands Table Of Contents

```
quota
    quota prints the users quota, it works like
    this: quota [arguments] <user/group>
    Where the arguments can be:

    -g      Print group quotas for the group of which the  user
            is a member.  The optional

    -u      flag is equivalent to the default.

    -v      will display quotas on filesystems where no storage
            is allocated.

    -q      Print a more terse message, containing only
            information on filesystems where usage is over quota.

    For more info on the quota command do: man quota
```

Index of Annotated Commands Table Of Contents

```
quotaoff
    quotaoff turns the quota off for a file system.
    quotaoff works like this: quotaoff [arguments] <filesystem>
    The arguments can be as follows:

    -a      Force  all file systems in /etc/fstab to have their
            quotas disabled.

    -v      Display a message for each file system affected.

    -u      Manipulate user quotas. This is the default.

    -g      Manipulate group quotas.
```

```
    This command is close to quotaon.

    For more info: man quotaon
    (Don't think there is a quotaoff man page, quotaon and quotaoff
    seems to have the same manual page)
```

Index of Annotated Commands Table Of Contents

```
quotaon
    quotaon turns the quota on for a file system.
    quotaon works like this: quotaon [arguments] <filesystem>
    The arguments can be as follows:

    -a      All file systems in  /etc/fstab  marked  read-write
            with quotas will have their quotas turned on.  This
            is normally used at boot time to enable quotas.

    -v      Display a message for each file system where quotas
            are turned on.

    -u      Manipulate user quotas. This is the default.

    -g      Manipulate group quotas.

    For more info: man quotaon
```

Index of Annotated Commands Table Of Contents

```
quotastats
    quotastats displays the quota stats .... can't find any help, --help
    or manual page for it.
```

Index of Annotated Commands Table Of Contents

```
read
    read, reads a variable.

    Example:

    echo -n "password: "
    read pass
    echo "Password was: $pass"

    For more info: help read
```

Index of Annotated Commands Table Of Contents

```
reboot
    reboot does what it says, it reboots the system, you have to be root
    to use this command.
    reboot works the same as: shutdown -r now
    or also the same as if you press: Ctrl+Alt+Del
    Nothing much more to say about the reboot command.
```

Index of Annotated Commands Table Of Contents

```
reset
    reset resets the console, say that you have accidentally done
    cat <binary-file> so you totally screwed up your console
    and you can't read anything on it, then just type `reset`
    and press enter, and it should be back to normal within some seconds.
```

Index of Annotated Commands Table Of Contents

```
rlogin
    remote login, if you wanna use this command do: man rlogin
    bore using it.
    The most common use of it is: rlogin -l <username> <host>
```

Index of Annotated Commands Table Of Contents

```
rm
    remove, remove/unlink files, rm can be used with the
```

```
    following arguments:

    -d, --directory        unlink directory, even if non-empty
                           (super-user only)
    -f, --force            ignore nonexistent files, never prompt
    -i, --interactive      prompt before any removal
    -r, -R, --recursive    remove the contents of directories recursively
    -v, --verbose          explain what is being done
        --help             display this help and exit
        --version          output version information and exit

    An example is, that if you have a directory called /foo
    that you wanna delete recursively, then you do: rm -rf /foo
    Or say that you have a file /foo/bar that you wanna remove
    without being prompted, then you do it like this: rm -f /foo/bar
```

Index of Annotated Commands Table Of Contents

```
rmmod
    remove modules, remove a loaded module.
    List the modules that you can remove with lsmod.
    And load modules with insmod.
    man any of them for more information.
```

Index of Annotated Commands Table Of Contents

```
route
    route, displays the routing table by default.
    The most common way of adding a route is like this:
    route add -host <ip> gw <other-ip>

    And to remove a post:
    route del -host <ip> gw <other-ip>

    An example would be, say that you want to route IP 123.123.123.123
    to 127.0.0.1, this would drop any connection attempts from
    123.123.123.123 to 127.0.0.1 so he can't connect to you or scan you,
    (this is true in most cases), you would do:

    route add -host 123.123.123.123 gw 127.0.0.1

    Now the route command is bigger then that, so if you wanna
    learn more about it do: man route
```

Index of Annotated Commands Table Of Contents

```
rpm
    rpm is a command that is very important to most distributions.
    rpm is short for 'redhat package manager' and was developed for
    RedHat by Caldera.
    rpm is a HUGE command, and works like this: rpm [arguments] <file>
    but here are the most commonly used arguments:

    rpm -ivh <package.rpm>  installs package.rpm
    rpm -Uvh <package.rpm>  updates package.rpm
    rpm -e <package>        un-installed/erases package
    rpm -qf <file>          displays what package the file came with
    rpm -qlp <package.rpm>  displays the contents of the package.rpm
    rpm -qRp <package.rpm>  displays the dependencies needed by package.rpm

    Other arguments and that are commonly used but not recommended are:

    --force         force install something
    --nodeps        do not check dependences

    Another thing is if you installed a *.src.rpm file (that ends up in
    /usr/src/RPM/*), you can compile a binary .rpm from it.
    Say that you installed some-package.src.rpm, then you would go to:
    /usr/src/RPM/SPECS/, and there type: rpm -ba some-package.spec
    wait a while during the compile, and then you would have a
    /usr/src/RPM/RPMS/<platform>/some-package.<platform>.rpm

    the "<platform>" is your platform, as i386, i486, i586, i686,
    k6, ppc, sprac, noarch etc.

    To create an rpm from a .src.rpm you first need to know that
    this should not be done as root for the simple reason that
    if you make an rpm as root several unworking parts of it may
    remain in your system generating errors if the compile of
```

```
    the rpm isn't successful.

    So the first thing you do to do this as user is to create a file
    named .rpmmacros
    And in that add the following:
    %_topdir        ~/RPM

    This should work to create the file:
    echo "%_topdir        $HOME/RPM" > ~/.rpmmacros

    Then you do this:
    mkdir -p ~/RPM/{SOURCES,SPECS,BUILD,RPMS,SRPMS}

    Now you're ready to start to build an rpm from a .src.rpm
    first (as user, not as root) install the source rpm.

    rpm -ivh package.src.rpm

    Then you go to ~/RPM/SPECS/

    The .src.rpm should have installed the sources in ~/RPM/SOURCES
    and the spec file in ~/RPM/SPECS/
    The spec file is like a script file, it tells rpm how to compile
    the source and build the rpm.

    Now find the spec file in ~/RPM/SPECS/, it's usually named the same
    as the package, like this:

    package.spec

    So not to make an rpm out of it, do this:

    rpm -ba package.spec

    If this is successful (which it sadly enough isn't every time because
    of ill written spec files)
    You should now have an rpm file in ~/RPM/RPMS/<your architecture>

    If you have a Pentium 2, the arch command will show "i586" and
    so the rpm will be found in ~/RPM/RPMS/i586/

    You will also have a brand new .src.rpm in ~/RPM/SRPMS/

    If you need to do the rpm to any other target than your own
    architecture, say you want to do it for i386, then you may do:

    rpm -ba package.spec --target=i386

    And so the new rpm will be found in ~/RPM/RPMS/i386/
    This is about all there is to say about the rpm command in this
    tutorial.
    The rpm command and the spec file *scripting* language
    would need a rather large tutorial by it self to be explained
    in full .... so I won't take up all that here.

    For more info on the rpm command do: man rpm
```

Index of Annotated Commands Table Of Contents

```
sed
    sed, stream editor, is already briefly explained in this tutorial,
    so if you want more info do: man sed
```

Index of Annotated Commands Table Of Contents

```
setleds
    setleds may show or set the flags and lights on NumLock, CapsLock
    and ScrollLock.
    On its own without any arguments it shows the current settings.

    The syntax is this: setleds [arguments] <+/-num,caps,scroll>

    Here's the arguments:

    -F      This  is the default. Only change the VT flags (and
            their setting may  be  reflected  by  the  keyboard
            leds).

    -D      Change both the VT flags and their default settings
            (so that a  subsequent  reset  will  not  undo  the
```

```
        change). This might be useful for people who always
        want to have numlock set.

-L      Do not touch the VT  flags,  but  only  change  the
        leds.  From this moment on, the leds will no longer
        reflect the VT flags (but display whatever  is  put
        into them). The command setleds -L (without further
        arguments) will restore the situation in which  the
        leds reflect the VT flags.

-num +num
        Clear  or  set  NumLock.   (At present, the NumLock
        setting influences  the  interpretation  of  keypad
        keys.   Pressing  the  NumLock  key complements the
        NumLock setting.)

-caps +caps
        Clear or set CapsLock.  (At present,  the  CapsLock
        setting  complements  the Shift key when applied to
        letters.  Pressing the CapsLock key complements the
        CapsLock setting.)

-scroll +scroll
        Clear or set ScrollLock.  (At present, pressing the
        ScrollLock key (or ^S/^Q) stops/starts console out-
        put.)

Here is a few example, where the first one is from the manual page,
(I'd hate to break the cut'n'paste tradition from the manual pages
now), so here are some examples:

      INITTY=/dev/tty[1-8]
      for tty in $INITTY; do
             setleds -D +num < $tty
      done

This would set numlock on for tty1 to tty8

Here's another short example:

      while /bin/true; do
            setleds -L +caps; usleep 500000
            setleds -L +num; usleep 500000
            setleds -L -caps; usleep 500000
            setleds -L -num; usleep 500000
      done

This would flash the NumLock and CapsLock leds, for infinity.

For more info do: man setleds
```

Index of Annotated Commands Table Of Contents

```
seq
    sequence numbers.
    seq works basically like this:

    seq [OPTION] LAST
    seq [OPTION] FIRST LAST
    seq [OPTION] FIRST INCREMENT LAST

    And can be used with the following options:

    -f, --format FORMAT      use printf(3) style FORMAT (default: %g)
    -s, --separator STRING   use STRING to separate numbers (default: \n)
    -w, --equal-width        equalize width by padding with leading zeroes
    --help                display this help and exit
    --version             output version information and exit

    Here's some small examples and what they do:

    seq 10          (Count from 1 to 10)
    seq 5 10        (Count from 5 to 10)
    seq 1 2 10      (Count from 1 to 10 by incrementing two: 1,3,5,7,9)
    seq 10 0        (Count backwards from 10 to 0)

    For more info do: seq --help
```

Index of Annotated Commands Table Of Contents

```
sleep
    sleep works like this: sleep <number of seconds>
    Not much to say about this command, ... if you wanna read
    more about it: man sleep
```

Index of Annotated Commands Table Of Contents

```
sort
    sort, sorts the contents of a file and gives the output to stdout.
    By default it sorts it in alphabetical order, sort works
    like this: sort [arguments] <file>

    sort can be executed with the following arguments:

    -b      ignore leading blanks in sort fields or keys

    -c      check if given files already sorted, do not sort

    -d      consider only [a-zA-Z0-9 ] characters in keys

    -f      fold lower case to upper case characters in keys

    -g      compare according to general numerical value, imply -b

    -i      consider only [ 40- 176] characters in keys

    -k POS1[,POS2]
            start a key at POS1, end it *at* POS2
            field  numbers  and  character offsets are numbered
            starting with one (contrast  with  zero-based  +POS form)

    -m      merge already sorted files, do not sort

    -M      compare (unknown) < `JAN' < ... < `DEC', imply -b

    -n      compare  according to string numerical value, imply -b

    -o FILE
            write result on FILE instead of standard output

    -r      reverse the result of comparisons

    -s      stabilize sort by disabling last resort comparison

    -t SEP  use SEParator instead of non- to whitespace transition

    -T DIRECTORY
            use  DIRECTORY  for temporary files, not $TMPDIR or /tmp

    -u      with -c, check for strict ordering; with  -m,  only
            output the first of an equal sequence

    -z      end  lines  with  0  byte,  not  newline,  for find -print0

    --help          display the help and exit

    --version       output version information and exit

    One more time I give thanks to the cut & paste function.

    Here's an example of sort: sort file1 -o sorted-file2
    This command works good with the `uniq` command to sort out
    duplicate words, like this: sort file1 | uniq > sorted-file

    For more info do: man sort
```

Index of Annotated Commands Table Of Contents

```
ssh
    secure shell, works a bit like telnet but has encryption,
    ssh is becoming a good standard of encrypted remote shell connections.
    ssh is however not usually default included in any distros,
    and there is several versions of it, so if you download it
    make sure to read all documentations about it.
    Even though it's not default included, I still wanted to include it
    in this tutorial to make users that use LAN connections
    as local networks with more than one user or cable modems aware
    of this tool, because if they use telnet anyone on the local
    subnet can sniff the connection and get any login and password
```

```
    used with incoming or outgoing telnet connections.
    Really anyone can sniff anything that is not encrypted, like
    ftp logins and passwords, http, IRC, and everything like that.
    but the most vital to protect is the ways people can enter
    your system, so if you are on a LAN with more than one user
    or have any form of cable or non-dialup connection,
    then disable telnet (put a # in front of the telnet line in
    /etc/initd.conf and after that do: killall -HUP initd), and then
    install ssh.
```

Index of Annotated Commands Table Of Contents

```
strip
    strip strips binary files (executables) of junk code,
    such as debugging information.
    This may be very useful to bring down the size of executable files.
    BUT beware, if you strip the kernel or any other very complex
    binary, they are likely to malfunction, so use this command
    wisely, and read its manual page.
```

Index of Annotated Commands Table Of Contents

```
su
    su, the manual pages says substitute user and the UNIX command bible
    says super user ... so it means any of those, it's however used
    to *become another user*, if you are root and su <user> you won't
    need to supply any password.
    If you type only `su` as user you will become root if you have the
    appropriate root password.
    su can be used with the following arguments:

    -, -l, --login               make the shell a login shell
    -c, --commmand=COMMAND       pass a single COMMAND to the shell with -c
    -f, --fast                   pass -f to the shell (for csh or tcsh)
    -m, --preserve-environment   do not reset environment variables
    -p                           same as -m
    -s, --shell=SHELL            run SHELL if /etc/shells allows it
        --help                   display this help and exit
        --version                output version information and exit

    Say now that you wanna su to root and have root's path/environment.
    then you do: su -
    Or say that you wanna execute a single command as root from being
    a user, say the command `adduser`, then you do: su -c "adduser"
    you will be prompted for the password, and if you can supply it
    the command will be executed as root.
```

Index of Annotated Commands Table Of Contents

```
swapoff
    turns swap off, it can be used with the following arguments:

    -h     Provide help
    -V     Display version
    -s     Display  swap usage summary by device.  This option
           is only available if /proc/swaps  exists  (probably
           not before kernel 2.1.25).
    -a     All  devices  marked  as  ``sw''  swap  devices  in
           /etc/fstab are made available.
    -p priority (man swapon and swapoff for more info on priorities)

    Example, say that you wanna turn all swap partitions (from /etc/fstab)
    off then you do: swapoff -a
    Again, for more info: man swapoff
```

Index of Annotated Commands Table Of Contents

```
swapon
    swapon is the opposite of swapoff but has the same arguments.
    See: man swapon
```

Index of Annotated Commands Table Of Contents

```
tail
    tail gives by default the last 10 lines out of a file, it's very alike
    the `head` command, and works like this: tail [arguments] <file>
    The most common usage of tail is this:
```

```
    tail -f <file>  This will append the data to stdout as the file grows.
                    very good to view logs as they come in.

    tail -50 <file> Displays the last 50 lines from a file.

    tail has more arguments which you can learn in its manual page
    if you are interested, do: man tail
```

Index of Annotated Commands Table Of Contents

```
talk
    talk is a little daemon controlled by inetd, so if it doesn't
    work on your local machine make sure the talk line in
    /etc/inetd.cond are not remmed by a leading # character.

    Talk gives a real time text chat, in a horizontally divided
    window or rather console.

    Talk works like this: talk user@host
    or just user if it's on the local machine.
    Say that I wanna send a talk request to user `alfa` on IP
    123.123.123.132, and I'm user `beta` on 234.234.234.234.
    Then I type: talk alfa@123.123.123.132
    And he as answer when the request comes
    types: talk beta@234.234.234.234

    What to type as answer comes up when you get a talk request.

    For more info on the talk command do: man talk
```

Index of Annotated Commands Table Of Contents

```
tar
    tar, UNIX tape archive, is yet another huge command,
    it's used to compress a directory to a compressed .tar file,
    or a single file to a tar file.
    tar works like this: tar [arguments] <directory-or-file>

    Here are the most common examples of tar usage:

    tar -zvxf <file.tar.gz> uncompress a .tar.gz or .tgz archive
    tar -vxf <file.tar>     uncompress a .tar archive
    tar -c --file=<file.tar> <directory>    crates a .tar archive
    tar -cf <file.tar> <directory>            - same as above -
    tar -tf <file.tar>      list the contents of a .tar file
    tar -tzf <file.tar.gz>  list the contents of a .tar.gz or a .tgz file
    tar -czvf <file.tar.gz> <directory>     crates a .tar.gz archive

    For more info on the tar command, do: man tar
```

Index of Annotated Commands Table Of Contents

```
tcpdump
    tcpdump is a command that let's you view the traffic on the local
    subnet or segment, It's not default on many Linux distributions.
    So if you have it or get it, read its documentation and its
    manual pages, if you want to use it.
```

Index of Annotated Commands Table Of Contents

```
telnet
    telnet is the most basic of all clients to know.
    It's not often you will ever use it in other ways than:
        telnet <host-or-ip> <port>
    And it's not even so often one uses it with the port number after.

    Telnet creates a real time connection to another computer,
    of course the other computer needs a running telnet daemon,
    and you need to have a login and a password to get in.
    But when you get in you can remotely work on the other system
    just as if you sat in front of it.

    Times when it's good to supply a port number after the host is
    most commonly to check the version of some daemon/server,
    as if you want to know the version of your own sendmail, you
    can always do: telnet 127.0.0.1 25
    smtp (send mail transfer protocol) runs on port 25.
```

```
    If you wonder what port something runs on check in /etc/services

    For more info on telnet do: man telnet
```

Index of Annotated Commands Table Of Contents

```
test
    test is a big command, and is used to generate boolean results
    out of 2 arguments, to explain the whole command here would take up
    to much space and time, it can be used like this:

    test -f /sbin/shutdown && echo "It's there" || echo "It's not there"

    That line says in clear English:
    test if file /sbin/shutdown is there, if outcome is true
    echo "It's there" else echo "It's not there".

    You can test if a file is executable, if a string is non-zero etc.
    Just about anything you can think of.

    For more info on the many things you can do with the `test`
    command, do: man test
```

Index of Annotated Commands Table Of Contents

```
touch
    touch will by default change the date on a file to the current date.
    It works like this: touch [arguments] <file>
    If the file doesn't exist if will create a file that is 0 bytes big.

    The following arguments can be used with touch:

    -a      change only the access time

    -c      do not create any files

    -d, --date=STRING
            parse STRING and use it instead of current time

    -f      (ignored)

    -m      change only the modification time

    -r, --reference=FILE
            use this file's times instead of current time

    -t     STAMP
            use [[CC]YY]MMDDhhmm[.ss] instead of current time

    --time=WORD
            access -a, atime -a, mtime -m, modify -m, use -a

    --help
            display the help and exit

    --version
            output version information and exit

    So say that you have a file called 'file' that I want to change date of
    to say 'Aug 21 1999 04:04',
    then you would do: touch -t 9908210404 file

    For more info on this command do: man touch
```

Index of Annotated Commands Table Of Contents

```
tr
    translate characters, this command can change all upper case
    characters to lower case characters in a file or substitute
    all numbers to some other characters etc.
    tr's syntax is: tr [arguments] <SET1> <SET2>
    tr can be used with the following arguments:

    -c, --complement
            first complement SET1

    -d, --delete
            delete characters in SET1, do not translate
```

```
    -s, --squeeze-repeats
            replace sequence of characters with one

    -t, --truncate-set1
            first truncate SET1 to length of SET2

    --help
            display this help and exit

    --version
            output version information and exit

    And the SET's are as follows:

    [:alnum:]
            all letters and digits

    [:alpha:]
            all letters

    [:blank:]
            all horizontal whitespace

    [:cntrl:]
            all control characters

    [:digit:]
            all digits

    [:graph:]
            all printable characters, not including space

    [:lower:]
            all lower case letters

    [:print:]
            all printable characters, including space

    [:punct:]
            all punctuation characters

    [:space:]
            all horizontal or vertical whitespace

    [:upper:]
            all upper case letters

    [:xdigit:]
            all hexadecimal digits

    [=CHAR=]
            all characters which are equivalent to CHAR

    Examples of tr is:
    cat file | tr [:upper:] [:lower:]       (change all uppercase to lower)
    cat file | tr -d [:alnum:]              (delete all numbers and chars)

    For more info on tr, do: man tr
```

Index of Annotated Commands Table Of Contents

```
traceroute
    traceroute is a command that traces a route to an IP/host
    and will give you the number of hops from your computer
    to the remote computer, and will display the ping times
    to each computer in the way.

    traceroute has some arguments that I never needed to use
    but if you feel curious about this command, feel free
    to look at the manual pages for it: man traceroute
```

Index of Annotated Commands Table Of Contents

```
ulimit
    ulimit sets a limit for how much memory etc. users are allowed
    to use.
    It works like this: ulimit [arguments]
    And the arguments can be the following:

    -S     use the `soft' resource limit
```

```
    -H     use the `hard' resource limit
    -a     show all settings
    -c     core file size (in blocks)
    -d     data seg size (in kilo bytes)
    -f     file size (in blocks)
    -l     max locked memory (in kilo bytes)
    -m     max memory size (in kilo bytes)
    -n     open files (number)
    -p     pipe size (512 bytes)
    -s     stack size (in kilo bytes)
    -t     cpu time (in seconds)
    -u     max user processes (number)
    -v     virtual memory (kilo bytes)

    Say that I want to set a limit that users can only run 50
    processes each, the I would do: ulimit -u 50
```

Index of Annotated Commands Table Of Contents

```
umount
    un mount, un mounts a mountpoint, say that you have mounted your
    CD-rom drive on /mnt/cdrom then you would do: umount /mnt/cdrom
    to unmount it.

    I never used any arguments to this command, but if you wanna learn
    about them, feel free to do: man umount
```

Index of Annotated Commands Table Of Contents

```
unalias
    unalias removes a defined alias, say that you have an alias
    like this: alias du='du -h'
    And you want to remove it: then you simply do: unalias du
    To remove all aliases do: unalias -a

    For more info do: help unalias
```

Index of Annotated Commands Table Of Contents

```
uname
    uname gives info on the current system, and works as
    this: uname [arguments]
    The arguments can be the following:

    -a, --all               print all information
    -m, --machine           print the machine (hardware) type
    -n, --nodename          print the machine's network node hostname
    -r, --release           print the operating system release
    -s, --sysname           print the operating system name
    -p, --processor         print the host processor type
    -v                      print the operating system version
    --help                  display this help and exit
    --version               output version information and exit

    The most common way of using uname is: uname -a
```

Index of Annotated Commands Table Of Contents

```
uncompress
    uncompress uncompresses .Z files, for more info do: man uncompress
```

Index of Annotated Commands Table Of Contents

```
uniq
    uniq does by default take away duplicate words out of a text,
    which can be good if your sorting out a dictionary.
    But `uniq` can also be executed with the following arguments:

    -c, --count
            prefix lines by the number of occurrences

    -d, --repeated
            only print duplicate lines

    -D, --all-repeated
            print all duplicate lines
```

```
    -f, --skip-fields=N
            avoid comparing the first N fields

    -i, --ignore-case
            ignore differences in case when comparing

    -s, --skip-chars=N
            avoid comparing the first N characters

    -u, --unique
            only print unique lines

    -w, --check-chars=N
            compare no more than N characters in lines

    -N      same as -f N

    +N      same as -s N

    --help
            display the help and exit

    --version
            output version information and exit

    For more info on this command do: man uniq
```

Index of Annotated Commands Table Of Contents

```
unset
    this command will remove an alias or function.
    It has the following options:

    -v      unset a variable only.
    -f      unset a function only.

    By default unset will first try to unset as a variable and if that
    fails it will try to unset as a function.

    Here's an example:

    alien:~$ foo=bar
    alien:~$ echo $foo
    bar
    alien:~$ unset foo
    alien:~$ echo $foo

    alien:~$

    For more info, do: help unset
```

Index of Annotated Commands Table Of Contents

```
unzip
    unzip is the tool or command to unzip files, it works like this:
    unzip [arguments] <file.zip>

    unzip has some arguments I never used, do: unzip --help
    to get a list of valid arguments.
    Also feel free to do: man unzip
```

Index of Annotated Commands Table Of Contents

```
updatedb
    update the locate database, updatedb works like this:
    updatedb [arguments] <pattern>
    By default updatedb updates the locate database so it covers the
    whole system and all its files, but has the following arguments:

    -u      Create slocate database starting at the root direc-
            tory.  This is the default behavior when called  as updatedb.

    -U path
            Create slocate database starting at path path.

    -e dirs
            Exclude  directories  in  the  comma-separated list
            dirs from the slocate database.
```

```
    -f fstypes
            Exclude file systems in  the  comma-separated  list
            dirs from the slocate database.

    -l <num>
            Security  level.   -l  0 turns security checks off,
            which will make searches faster.  -l 1 turns  security
            checks on. This is the default.

    -q      Quiet mode; error messages are suppressed.

    -v      Verbose  mode;  display files indexed when creating database

    --help
            Print a summary of the options to slocate and exit.

    --version
            Print the version number of slocate and exit.

    If you have the whole system updated in the locate database,
    to find a file all you have to do is to: locate <file>

    For more info: man updatedb
```

Index of Annotated Commands Table Of Contents

```
uptime
    displays the current uptime (the time the system has been on).
    with the load average.
    It shows from left to right:
    The current time, how long the system has been running,
    how many users are currently logged on, and the system load
    averages for the past 1, 5, and 15 minutes.

    Also do: man uptime
```

Index of Annotated Commands Table Of Contents

```
useradd
    useradd adds a user account to the system.
    useradd works like this: useradd [arguments] user

    Here's a cut & paste from its manual page (as usual).

    -c comment
            The new user's password file comment field.

    -d home_dir
            The  new user will be created using home_dir as the
            value for the user's login directory.  The  default
            is to append the login name to default_home and use
            that as the login directory name.

    -e expire_date
            The date on which the user  account  will  be  dis-
            abled.   The  date is specified in the format YYYY-
            MM-DD.

    -f inactive_days
            The number of days after a password  expires  until
            the  account is permanently disabled.  A value of 0
            disables the account as soon as  the  password  has
            expired,  and  a  value of -1 disables the feature.
            The default value is -1.

    -g initial_group
            The group name or  number  of  the  user's  initial
            login  group.   The group name must exist.  A group
            number must refer to  an  already  existing  group.
            The default group number is 1.

    -G group,[...]
            A  list  of  supplementary groups which the user is
            also a member of.  Each group is separated from the
            next  by  a  comma, with no intervening whitespace.
            The groups are subject to the same restrictions  as
            the group given with the -g option.  The default is
            for the user to belong only to the initial group.

    -m      The user's home directory will  be  created  if  it
```

```
          does  not  exist.   The  files  contained in skele-
          ton_dir will be copied to the home directory if the
          -k option is used, otherwise the files contained in
          /etc/skel will be used  instead.   Any  directories
          contained in skeleton_dir or /etc/skel will be cre-
          ated in the user's home directory as well.  The  -k
          option  is  only  valid  in conjunction with the -m
          option.  The default is to not create the directory
          and to not copy any files.

    -M    The  user  home directory will not be created, even
          if the system wide settings from /etc/login.defs is
          to create home dirs.

    -n    A  group  having  the  same  name as the user being
          added to the system will  be  created  by  default.
          This  option  will turn off this Red Hat Linux spe-
          cific behavior.
    -r    This flag is used to create a system account.  That
          is, an user with an UID lower than value of UID_MIN
          defined in /etc/login.defs. Note that useradd  will
          not  create  a  home  directory  for  such an user,
          regardless    of    the    default    setting    in
          /etc/login.defs.   You have to specify -m option if
          you want a home directory for a system  account  to
          be created.  This is an option added by Red Hat.

    -p passwd
          The  encrypted  password,  as returned by crypt(3).
          The default is to disable the account.

    -s shell
          The name of the user's login shell.  The default is
          to  leave this field blank, which causes the system
          to select the default login shell.

    -u uid  The numerical value of the user's ID.   This  value
          must  be unique, unless the -o option is used.  The
          value must be non-negative.  The default is to  use
          the  smallest  ID value greater than 99 and greater
          than every other user.  Values between 0 and 99 are
          typically reserved for system accounts.


    When the -D argument is used useradd with either give the
    default values or update them if there is more arguments.
    The other arguments can be:

    -b default_home
          The  initial  path  prefix  for  a  new user's home
          directory.  The user's name will be affixed to  the
          end  of  default_home  to  create the new directory
          name if the -d option is not used when  creating  a
          new account.

    -e default_expire_date
          The date on which the user account is disabled.

    -f default_inactive
          The  number  of  days  after a password has expired
          before the account will be disabled.

    -g default_group
          The group name or  ID  for  a  new  user's  initial
          group.  The named group must exist, and a numerical
          group ID must have an existing entry .

    -s default_shell
          The name of the new user's login shell.  The  named
          program  will  be  used  for  all  future  new user
          accounts.

    Also feel free to read the manual page: man useradd
```

Index of Annotated Commands Table Of Contents

```
userdel
    userdel removes a user from the system.
    userdel works like this: userdel [argument] <user>
    The only argument to this command is:
    -r    removes the users home directory, along with the user.
```

```
    This will delete the users login and everything from the system.

    userdel will not remove the user if he is currently logged
    in to the system or have any processes running.
    So make sure you kill all processes owned by the user,
    if any, before removing his/her account.

    To kill the all running processes owned by the same user
    you can do the following command (change <user> to the username):

    for pids in `ps U <user> | cut -c 1-6`; do kill -9 $pids ; done
```

Index of Annotated Commands Table Of Contents

```
users
    Display the currently logged in users.
```

Index of Annotated Commands Table Of Contents

```
usleep
    usleep is another version of the sleep command, but instead of being
    told how many seconds to sleep, it sleeps in microseconds.
    So `usleep 1000000` makes it sleep for 1 second.
```

Index of Annotated Commands Table Of Contents

```
w
    w is like a mix of who and finger, it's used to see who's logged
    on to the system and will show the following:
    login name, terminal, host, login time, idle time, JCPU (total cpu time
    that user (terminal) takes up), PCPU (cpu time of the users current
    process which is shown in the next field), what (process)
```

Index of Annotated Commands Table Of Contents

```
wall
    wall is a superuser command to send a string of text to all
    consoles/terminals, wall can work either like: wall <string>
    or: wall <file with string in it>
    To wall special characters like "=)" you need to do it like
    this: wall '<string> =)'
```

Index of Annotated Commands Table Of Contents

```
wc
    word count, works basically like this: wc [argument] <file>
    Where the arguments can be any of the following:

    -c, --bytes, --chars   print the byte counts
    -l, --lines            print the newline counts
    -L, --max-line-length  print the length of the longest line
    -w, --words            print the word counts
        --help             display this help and exit
        --version          output version information and exit

    So to find out the number of words in a file called say "file1",
    you would do: wc -w file1
    Or to find out the number of lines in the same file you would
    do: wc -l file1

    This little tool can prove to be very useful, though when you
    use it with the -l/--lines argument it will only count lines that
    contains any characters, if you wanna count all lines
    including empty lines, use: grep -c . file1
    or the same thing in another way: cat file1 | grep -c .
```

Index of Annotated Commands Table Of Contents

```
whatis
    whatis searches for words in the whatis database, say that you
    are looking for a manual page about the `shutdown` command.
    Then you can do: whatis shutdown
    for more information, do: man whatis
    Or: man apropos
```

Index of Annotated Commands Table Of Contents

```
whereis
    whereis looks for something just as the `which` command here below.
    but looks for matches in more places, as the manual page directory's.
    It works like this: whereis [argument] <what-you-wanna-find>
    Try this command a few times, and if you want to learn
    more about it as its arguments and so do: man whereis
```

Index of Annotated Commands Table Of Contents

```
which
    which will tell you where a command is located, as if you
    do: which shutdown
    it will answer: /sbin/shutdown
    This command will search your path for whatever you type after it.
    This command is best used in scripts and functions, like this:

    function whichrpm { rpm -qf `which $1`; }

    The which command has some arguments, and more examples in its
    manual page, so for more info do: man which
```

Index of Annotated Commands Table Of Contents

```
who
    who is a little command that shows you who's logged on,
    on what tty and at what time they logged on.
    I never ever used this command with any arguments,
    but if you want to learn more about this command do: who --help
    or: man who
```

Index of Annotated Commands Table Of Contents

```
whoami
    displays your user name, same as `id -un`.
    whoami can be used with the following arguments:
    --help      display this help and exit
    --version   output version information and exit

    Not much more to say about this command.
```

Index of Annotated Commands Table Of Contents

```
whois
    whois is a tool that asks internic for information on a domain name.
    This is only for .com .org .net etc.
    If any luck it will give you the name and other info of the one that
    registered the domain, and its name servers.

    whois can also be used to do: whois user@host

    For more info on this command do: man whois
    Or: man fwhois
```

Index of Annotated Commands Table Of Contents

```
yes
    yes is a command that repeats the same thing over and over again,
    it's used as this: yes <string>
    If no string or word is supplied it will repeat the character 'y'.
    yes can be used with the following arguments:

    --version       display the version and exit.
    --help          display the help and exit.
```

Index of Annotated Commands Table Of Contents

```
zip
    zip is a compression tool, to compress with zip do:

    zip [arguments] <file.zip> <file1> <file2> etc.

    The arguments can be the following:
```

```
    -f   freshen: only changed files
    -u   update: only changed or new files
    -d   delete entries in zipfile
    -m   move into zipfile (delete files)
    -r   recurse into directories
    -j   junk (don't record) directory names
    -0   store only
    -l   convert LF to CR LF (-ll CR LF to LF)
    -1   compress faster
    -9   compress better
    -q   quiet operation
    -v   verbose operation/print version info
    -c   add one-line comments
    -z   add zipfile comment
    -@   read names from stdin
    -o   make zipfile as old as latest entry
    -x   exclude the following names
    -i   include only the following names
    -F   fix zipfile (-FF try harder)
    -D   do not add directory entries
    -A   adjust self-extracting exe
    -J   junk zipfile prefix (unzipsfx)
    -T   test zipfile integrity
    -X   eXclude eXtra file attributes
    -y   store symbolic links as the link instead of the referenced file
    -R   PKZIP recursion (see manual)
    -h   show the help
    -n   don't compress these suffixes

    To uncompress a zip file, use the `unzip` command.
```

Index of Annotated Commands Table Of Contents

And that is most of the commands you'll ever encounter while scripting or using a *nix system.

There are LOADS of other commands, but not many that are as frequently used as these I just explained.

There are more really useful commands that I never seen as default on any system as well, like `pgp` and `gpg`, I haven't included those since there full documentation comes with the same package as that command/application if you download it.

So, as I said these are the most useful commands, but if someone out there think I missed some really useful command send me a mail and I'll add it.

Index of Annotated Commands Table Of Contents

End of document