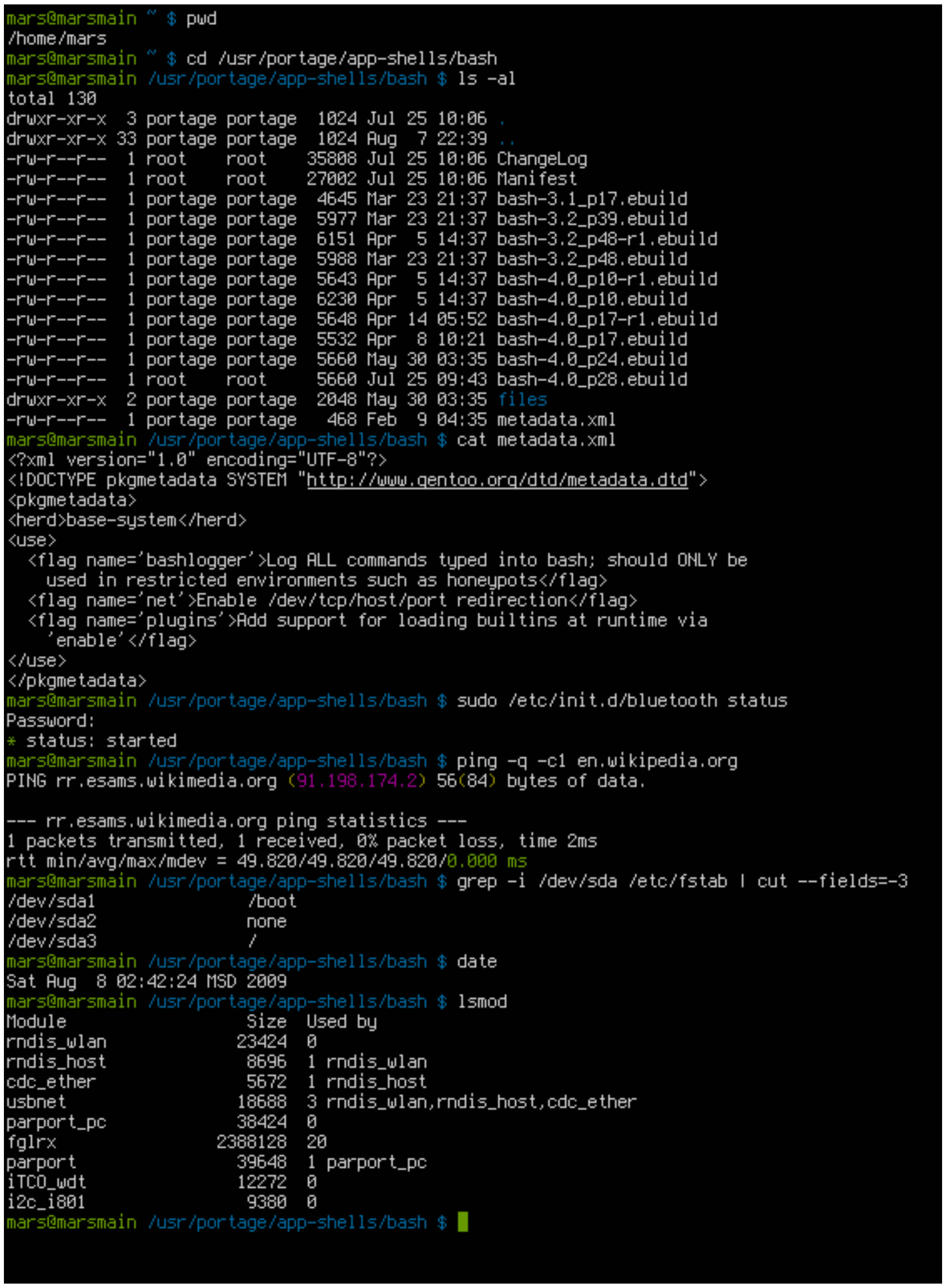
BASH

Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell. First released in 1989, it has been distributed widely as the default login shell for Linux distributions and Apple's macOS.



```
mars@marsmain ~ $ pwd
/home/mars
mars@marsmain ~ $ cd /usr/portage/app-shells/bash
mars@marsmain /usr/portage/app-shells/bash $ ls -al
total 130
drwxr-xr-x  3 portage portage  1024 Jul 25 10:06 .
drwxr-xr-x 33 portage portage  1024 Aug  7 22:39 ..
-rw-r--r--  1 root    root    35808 Jul 25 10:06 ChangeLog
-rw-r--r--  1 root    root    27002 Jul 25 10:06 Manifest
-rw-r--r--  1 portage portage  4645 Mar 23 21:37 bash-3.1_p17.ebuild
-rw-r--r--  1 portage portage  5977 Mar 23 21:37 bash-3.2_p39.ebuild
-rw-r--r--  1 portage portage  6151 Apr  5 14:37 bash-3.2_p48-r1.ebuild
-rw-r--r--  1 portage portage  5988 Mar 23 21:37 bash-3.2_p48.ebuild
-rw-r--r--  1 portage portage  5643 Apr  5 14:37 bash-4.0_p10-r1.ebuild
-rw-r--r--  1 portage portage  6230 Apr  5 14:37 bash-4.0_p10.ebuild
-rw-r--r--  1 portage portage  5648 Apr 14 05:52 bash-4.0_p17-r1.ebuild
-rw-r--r--  1 portage portage  5532 Apr  8 10:21 bash-4.0_p17.ebuild
-rw-r--r--  1 portage portage  5660 May 30 03:35 bash-4.0_p24.ebuild
-rw-r--r--  1 root    root     5660 Jul 25 09:43 bash-4.0_p28.ebuild
drwxr-xr-x  2 portage portage  2048 May 30 03:35 files
-rw-r--r--  1 portage portage   468 Feb  9 04:35 metadata.xml
mars@marsmain /usr/portage/app-shells/bash $ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
<herd>base-system</herd>
<use>
  <flag name='bashlogger'>Log ALL commands typed into bash; should ONLY be
    used in restricted environments such as honeypots</flag>
  <flag name='net'>Enable /dev/tcp/host/port redirection</flag>
  <flag name='plugins'>Add support for loading builtins at runtime via
    'enable'</flag>
</use>
</pkgmetadata>
mars@marsmain /usr/portage/app-shells/bash $ sudo /etc/init.d/bluetooth status
Password:
 * status: started
mars@marsmain /usr/portage/app-shells/bash $ ping -q -c1 en.wikipedia.org
PING rr.esams.wikimedia.org (91.198.174.2) 56(84) bytes of data.

--- rr.esams.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 49.820/49.820/49.820/0.000 ms
mars@marsmain /usr/portage/app-shells/bash $ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1               /boot
/dev/sda2               none
/dev/sda3               /
mars@marsmain /usr/portage/app-shells/bash $ date
Sat Aug  8 02:42:24 MSD 2009
mars@marsmain /usr/portage/app-shells/bash $ lsmod
Module                  Size  Used by
rndis_wlan             23424  0
rndis_host              8696  1 rndis_wlan
cdc_ether               5672  1 rndis_host
usbnet                 18688  3 rndis_wlan,rndis_host,cdc_ether
parport_pc             38424  0
fglrx                2388128  20
parport                39648  1 parport_pc
iTCO_wdt               12272  0
i2c_i801                9380  0
mars@marsmain /usr/portage/app-shells/bash $ █
```

Linux File Permissions

Basic File Permissions

Permission Groups

Each file and directory has three user based permission groups:

- **Owner(o)** - The Owner permissions apply only the owner of the file or directory, they will not impact the actions of other users.
- **Group(g)** - The Group permissions apply only to the group that has been assigned to the file or directory, they will not effect the actions of other users.
- **All users(u)** - The All Users permissions apply to all other users on the system, this is the permission group that you want to watch the most.

Permission Types

Each file or directory has three basic permission types:

- **Read(r)** - The Read permission refers to a user's capability to read the contents of the file.
- **Write(w)** - The Write permissions refer to a user's capability to write or modify a file or directory.
- **Execute(x)** - The Execute permission affects a user's capability to execute a file or view the contents of a directory.

Advanced Permissions

The special permissions flag can be marked with any of the following:

- _ - no special permissions
- d - directory
- l- The file or directory is a symbolic link
- s - This indicated the setuid/setgid permissions. This is not set displayed in the special permission part of the permissions display, but is represented as a s in the read portion of the owner or group permissions.
- t - This indicates the sticky bit permissions. This is not set displayed in the special permission part of the permissions display, but is represented as a t in the executable portion of the all users permissions

Linux Filesystem Overview

~	Special shortcut, refers to the current user's home directory
/	Root directory

Absolute Path

An *absolute path name* starts with the character / (identifying the *root directory*, which contains all other directories and files), then every child directory that must be traversed to reach the element is listed, each separated by a / sign.

/usr/bin

/var/www/index.html

/etc/resolv.conf

Relative Path

A *relative path name* is one that doesn't start with /; in that case, the directory tree is traversed starting from a given point, which changes depending on context, called the *current directory*. In every directory, there are two special directories:

.	refers to the directory itself (current directory)
..	refers to parent directory

Main directories

/bin is a place for most commonly used terminal commands, like ls, mount, rm, etc.

/dev contains all device files, which are not regular files but instead refer to various hardware devices on the system, including hard drives.

/etc contains system-global configuration files, which affect the system's behavior for all users.

/home is the place for users' home directories.

/lib contains very important dynamic libraries and kernel modules

/mnt is also a place for mount points, but dedicated specifically to "temporarily mounted" devices, such as network filesystems.

/proc is a virtual filesystem that provides a mechanism for kernel to send information to processes.

/root is the superuser's home directory, not in /home/ to allow for booting the system even if /home/ is not available.

/sbin contains important administrative commands that should generally only be employed by the superuser.

/sys is a virtual filesystem that can be accessed to set or obtain information about the kernel's view of the system.

/tmp is a place for temporary files used by applications.

/usr contains the majority of user utilities and applications, and partly replicates the root directory structure, containing for instance, among others, /usr/bin/ and /usr/lib.

/var is dedicated to variable data, such as logs, databases, websites, and temporary spool (e-mail etc.) files that persist from one boot to the next. A notable directory it contains is /var/log where system log files are kept.

BASH commands

ls
Lists the contents of a directory. Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

cd
The cd command is one of the commands you will use the most at the command line in linux. It allows you to change your working directory. You use it to move around within the hierarchy of your file system.

mv

The mv command is used to move or rename files. mv renames file SOURCE to DEST, or moves the SOURCE file (or files) to DIRECTORY.

cp
The cp command is used to make copies of files and directories.

scp
Copy files or directory from or to remote host

man
The man command - the manual command - is used to show the manual of the inputted command. Inputting the man command will show you all information about the command you are using. For example:

```
man cd
```

this command will show the manual or all relevant information for the change directory command.

mkdir

The mkdir - make directory - command allows the user to make a new directory. Just like making a new directory within a PC or Mac desktop environment, the mkdir command makes new directories in a Linux environment. An example of the mkdir command

```
mkdir testdirectory
```

The example command made the directory "testdirectory".

touch

The touch command - a.k.a. the make file command - allows users to make files using the Linux CLI. Just as the mkdir command makes directories, the touch command makes files. Just as you would make a .doc or a .txt using a PC desktop, the touch command makes empty files. An example of the touch command:

```
touch testfile.txt
```

The example touch command effectively created the file testfile.txt. As noted by the extension, the file created is a .txt or text file. To equate, a .txt file in Linux is akin to a .txt notebook file within a Windows or Mac OS.

rm

Remove file or directory

grep

Print lines matching a pattern.

grep searches the named input FILEs (or **standard input** if no files are named, or if a single hyphen-minus (-) is given as file name) for lines containing a match to the given pattern. By default, grep prints the matching lines.

```
cat foo.txt | grep bar
```

outputs all lines in data.txt that contains “bar”

root (*superuser*)

root is the user name or account that by default has access to all commands and files on a Linux or other Unix-like operating system. It is also referred to as the root account, root user and the superuser.

Root privileges are the powers that the root account has on the system. **The root account is the most privileged on the system and has absolute power over it** (i.e., complete access to all files and commands). Among root's powers are the ability to modify the system in any way desired and to grant and revoke access permissions (i.e., the ability to read, modify and execute specific files and directories) for other users, including any of those that are by default reserved for root.

sudo

Execute a single command as the superuser

Root privileges are usually required for installing software using package control software because of the need to write to system directories.

```
sudo rm –rf /                 #do not try this

sudo apt-get install links    #install the links package using the apt-get package control
```

su

Switch user, or becomes the superuser (when no arguments were provided)

List of common BASH commands

`clear`	clear all previous commands' output text from the terminal
`exit` (or `logout`)	quits the shell
`history`	show a list of all past commands you have typed into this shell

`ls`	list files in a directory
`pwd`	displays the shell's current working directory
`cd`	changes the shell's working directory to the given directory; can be a relative or absolute path
`mkdir`	creates a new directory with the given name
`rmdir`	removes the directory with the given name (the directory must be empty)

`cp`	copies a file/directory
`mv`	moves (or renames) a file/directory
`rm`	deletes a file
`touch`	update the last-modified time of a file (or create an empty file)

`cat`	output the contents of a file
`more` (or `less`)	output the contents of a file, one page at a time
`head`, `tail`	output the beginning or ending of a file
`wc`	output a count of the number of characters, lines, words, etc. in a file
`du`	report disk space used by a file/directory
`diff`	output differences between two files

`cat`	output the contents of a file
`more` (or `less`)	output the contents of a file, one page at a time
`head`, `tail`	output the beginning/ending of a file
`wc`	output a count of the number of characters, lines, words, etc. in a file
`du`	report disk space used by a file/directory
`diff`	output differences between two files

`whoami`	outputs your user name
`passwd`	changes your password
`groups`	list the groups to which a user belongs
`sudo`	execute a single command as the super-user
`su`	log in to a shell as the super-user

`nano`	crappy but simple text editors (recommended)
`vi`, `vim`	another complicated text editor (not recommended)

Shell (Bash) Script

A Bash script is a plain text file which contains a series of commands. These commands are a mixture of commands we would normally type ourselves on the command line (such as ls or cp for example) and commands we could type on the command line but generally wouldn't. An important point to remember though is:

Anything you can run normally on the command line can be put into a script and it will do exactly the same thing. Similarly, anything you can put into a script can also be run normally on the command line and it will do exactly the same thing.

You don't need to change anything. Just type the commands as you would normally and they will behave as they would normally. It's just that instead of typing them at the command line we are now entering them into a plain text file. In this sense, if you know how to do stuff at the command line then you already know a fair bit in terms of Bash scripting.

Running a shell script

```
./script.sh

sh script.sh

bash script.sh
```

Environment Variables

An environment variable is a named object that contains data used by one or more applications. In simple terms, it is a variable with a name and a value. The value of an environmental variable can for example be the location of all executable files in the file system, the default editor that should be used, or the system locale settings. Users new to Linux may often find this way of managing settings a bit unmanageable. However, environment variables provide a simple way to share configuration settings between multiple applications and processes in Linux.

To list the current environmental variables with values:

```
printenv
```

Setting an environmental variable

```
export <variable>=<path_to_directory>
```

System Variable	Meaning	To View Variable Value Type
HOSTNAME	The name of your computer.	echo $HOSTNAME
HOME	The home directory of the current user.	echo $HOME
LANG	Used to determine the locale category for any category not specifically selected with a variable starting with LC_.	echo $LANG
PATH	The search path for commands. It is a colon-separated list of directories in which the shell looks for commands.	echo $PATH
SHELL	Set path to login shell.	echo $SHELL

GCC (GNU Compiler Collection)

GCC, formerly for "*GNU C Compiler*", has grown over times to support many languages such as C++, Objective-C, Java, Fortran and Ada. It is now referred to as "*GNU Compiler Collection*". The mother site for GCC is http://gcc.gnu.org/.
GCC is a key component of "*GNU Toolchain*", for developing applications, as well as operating systems. The GNU Toolchain includes:

1. GNU Compiler Collection (GCC): a compiler suit that supports many languages, such as C/C++, Objective-C and Java.
2. GNU Make: an automation tool for compiling and building applications.
3. GNU Binutils: a suit of binary utility tools, including linker and assembler.
4. GNU Debugger (GDB).
5. GNU Autotools: A build system including Autoconf, Autoheader, Automake and Libtool.
6. GNU Bison: a parser generator (similar to lex and yacc).

G++ Compilation Process

Preprocessing

```
g++ source.cpp –E -o source.i
```

Compilation

```
g++ source.i –o source.s
```

Assemble

```
g++ -c source.s –o source.o
```

Linking

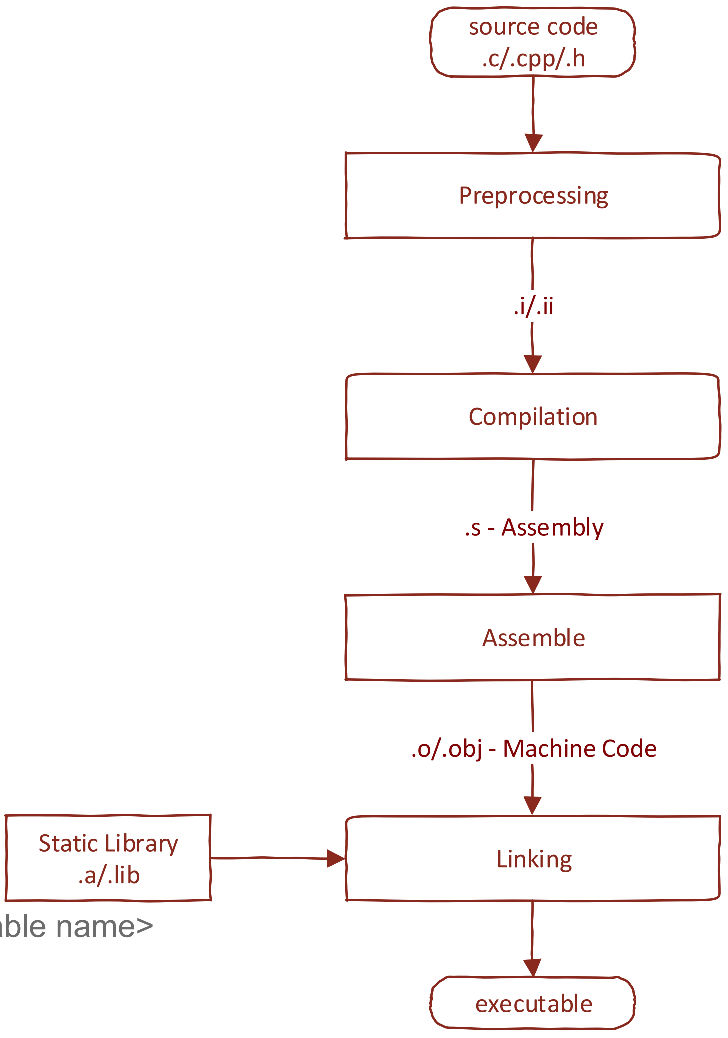
```
g++ source.o –o app
```

Usually, just

```
g++ source.cpp –o app
g++ source.cpp
```

Multiple source files

```
g++ <source1> <source2> … -o <executable name>
```



Makefile

The Makefile directs make on how to compile and link a program. Using C/C++ as an example, when a C/C++ source file is changed, it must be recompiled. If a header file has changed, each C/C++ source file that includes the header file must be recompiled to be safe. Each compilation produces an object file corresponding to the source file. Finally, if any source file has been recompiled, all the object files, whether newly made or saved from previous compilations, must be linked together to produce the new executable program. These instructions with their dependencies are specified in a Makefile. If none of the files that are prerequisites have been changed since the last time the program was compiled, no actions take place. For large software projects, using Makefiles can substantially reduce build times if only a few source files have changed.

Rules

A makefile consists of “rules” in the following form

```
target: dependencies
        command(s) …
```

A target is usually the name of a file that is generated by a program; examples of targets are executable or object files. A target can also be the name of an action to carry out, such as "clean".

A dependency (also called prerequisite) is a file that is used as input to create the target. A target often depends on several files. However, the rule that specifies a recipe for the target need not have any prerequisites. For example, the rule containing the delete command associated with the target "clean" does not have prerequisites.

The command(s) (also called recipe) is an action that make carries out. A recipe may have more than one command, either on the same line or each on its own line. Note the use of meaningful indentation in specifying commands; also note that the indentation must consist of a single <tab> character.

Executing a Makefile

Simply,

```
make
```