Linux Crash Course

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License. The short link to these notes is: <https://goo.gl/hYAqTL>

Authors: Carlisle Childress 2018

========================

Table of contents:

[**Key Unix concepts**](#_bv1z0np32zfk) **3**

[What is a shell?](#_72m8qxpmzovt) 3

[What is a terminal?](#_792esc688v1k) 3

[What is the environment?](#_9xcp95559cv5) 4

[What is the path?](#_2cxny15dvsdh) 4

[What is POSIX ?](#_7o73wugqkat2) 5

[Other Resources:](#_734movdjmj94) 5

[**How to find help**](#_q4i1kg4v6loh) **6**

[Google](#_tmd0t0bxe91g) 6

[Wikipedia](#_1pflun1c082i) 6

[Gnu](#_xmlyarxlf8qg) 6

[RedHat](#_yuejhalzave8) 6

[Manual pages](#_2o7g22wtf82o) 7

[Safari Books online](#_x9mjwpwsn3a) 7

[HowTos, Blogs, Wiki](#_ugnaz47g98mw) 8

[**Commands to handle files**](#_so199h237l05) **8**

[Beginning Commands: ls](#_af9zj9ebrp5j) 8

[What are parameters?](#_rdohfyejvtk3) 8

[What is a switch?](#_nva8o6upid0p) 9

[What are Standard Wildcards?](#_ejm4hwv8is3r) 9

[What is a "Magic Number?"](#_5rv2zyk83zlp) 10

[History](#_7p8qxzc2ls27) 10

[**Deleting**](#_o46hw3uwg0ce) **12**

[**Commands dealing with directories**](#_h1m1yeosfq9u) **13**

[**Commands dealing with processes**](#_4hozbho0q0zi) **13**

[Pipes and Redirection](#_r9s9mo4q41xm) 14

[**Commands to use across networks**](#_t8kb1fwaja07) **15**

[**Editing with vim**](#_naefg4dcx3pu) **16**

[Exiting vim](#_58fqz8iunu5s) 16

[Command mode](#_6isxx445eghu) 16

[Editing mode](#_t9knybjvhki6) 17

[Vimtutor](#_e0x63hkkyc3o) 17

[Advanced commands](#_pi9ong0hbkb) 17

[**Commands for handling strings: searching, cutting, replacing**](#_hl19k3qdrdcf) **18**

[find](#_x1smmcewymnc) 18

[grep](#_omt3l4uy8a2j) 19

[Regular Expressions](#_30xwbquvyjsv) 19

[sed awk cut](#_cczjr75qy9vv) 20

[**File archives, compression, and checksums**](#_wn2fyv67ffha) **21**

[tar](#_jfvh16w05u8e) 21

[Compression](#_unj3sg5i8tsq) 21

[checksums](#_t9vw7tv3y2nb) 22

[Md5deep](#_o1760bdf9u13) 22

[Signatures](#_635ux8w6h11j) 22

[**Bash Scripting**](#_hqdlltv0q5pt) **22**

[Shebang](#_4pmuzu9qf5gf) 23

[A simple bash script](#_7af6din0rno6) 23

[Bash variables](#_j8ylwad8tb4k) 24

[echo vs print](#_acerz8rvft4d) 24

[Special Variables: $0 $1 $2 $#](#_943m199nbq92) 24

[Where to find more bash help](#_yag0teu58ac9) 28

[**Batch processing with gridengine**](#_aam2zhlphbct) **28**

[**Batch Processing with slurm**](#_cxkzg5ptv5) **29**

[**Compiling Basics**](#_1v9d76pdnl9b) **30**

[GNU Build system](#_erthyi4hcs7g) 31

[**Putting it all together: A compilling example:**](#_3hd2c9meo6al) **32**

[Other early computer games: Adventure](#_w9oaavv5pz8t) 34

==========

<https://boyter.org/2016/04/collection-orly-book-covers/>

# Key Unix concepts

Linux is based upon design decisions used to create the Unix operating system

* Everything is a file

The advantage of this approach is that the same set of tools, utilities and APIs can be used on a wide range of resources.

<https://en.wikipedia.org/wiki/Everything_is_a_file>

* File names are case sensitive
* Files have owners and groups that limit who can use the file
* Files have privileges that allow or deny read, write and execution.

Other key concepts:

* Write programs that do one thing and do it well.
* Write programs to work together.
* Write programs to handle text streams, because that is a universal interface.

<https://en.wikipedia.org/wiki/Unix_philosophy>

## What is a shell?

A user interface provided by the operating system that allows commands to be issued to the operating system and sometimes includes scripting capabilities.

<https://en.wikipedia.org/wiki/Unix_shell>

The shell most commonly used under Linux is called bash. There are other shells as well. Other operating systems also have shells: Under Windows it's called the Command Prompt for older versions and Powershell on newer versions. OSX uses bash, just like Linux.  
  
If you are unsure which shell you are currently using, the following command will show the shell binary. Most of the time it will be bash:

echo $SHELL

## What is a terminal?

Originally it was hardware, a display and keyboard that allowed a user to interact with a computer. But today, it usually means a piece of software that acts like that hardware. The terminal is software used by the user to make a connection to a server.

<https://en.wikipedia.org/wiki/Terminal_emulator>

<http://askubuntu.com/questions/506510/what-is-the-difference-between-terminal-console-shell-and-command-line>

## What is the environment?

The unix/linux environment consists of a set of environmental variables: some set by the operating system, some by the shell, some by the user and some by applications or scripts.

<http://www.tutorialspoint.com/unix/unix-environment.htm>

Show the entire environment with this command:

env

One of the most important environmental variables is the path.

## What is the path?

The path is one of the most used variables in the environment. The path is an ordered list of directories to search for when executable is requested to run.

Show the contents of the PATH environmental variable with this command:

echo $PATH

How to append path

export PATH=/additional/path:$PATH

The path is usually appended to the front of the PATH. Separate directories must be separated by a colon. There is no space around the = and the must be a $ in front of the final PATH, but not the first PATH

The export command means that the variable is not only available to this shell, but to all shells that are spawned by this shell, also called child processes

Example:

$ FOO=bar  
 $ sh -c 'echo $FOO'  
  
 $ export FOO=barnone  
 $ sh -c 'echo $FOO'  
 bar

<https://stackoverflow.com/questions/7328223/unix-export-command>

How to determine if a given command in the path? Use the which command:

which <command>

also multiple commands:

which <command1> <command2> <command3>

Example: Append the path to include the blastx application that is located in the directory /usr/local/ncbi­-blast­-2.6.0+/bin

1. Show original path:

echo $PATH

2. Is the blastx application in this path?

which blastx

2. Append path

export PATH=/usr/local/ncbi-­blast-­2.6.0+/bin:$PATH

3. Show path again

echo $PATH

4. Try which again

which blastx

5. Show version of blastx

blastx -version

## What is POSIX ?

"The Portable Operating System Interface (POSIX) is a family of standards for maintaining compatibility between operating systems. POSIX defines the application programming interface (API), along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems."

<https://en.wikipedia.org/wiki/POSIX>

POSIX is why there is compatibility between different unix and linux operating systems. It also allows for some compatibility between linux, windows, osx and other operating systems.

## Other Resources:

<http://www.ee.surrey.ac.uk/Teaching/Unix/>

<http://cli.learncodethehardway.org/book/>

<http://cli.learncodethehardway.org/bash_cheat_sheet.pdf>

<http://www.cheatography.com/davechild/cheat-sheets/linux-command-line/>

<http://bioinformatics.mdc-berlin.de/intro2UnixandSGE/unix_for_beginners/README.html>

# How to find help

<https://i.imgur.com/fhgzVEt.jpg>

<https://imgur.com/gallery/vqUQ5>

<https://dev.to/rly>

## Google

We all know how to google, right?

## Wikipedia

Search with command name + linux and it will show you references.

example: google: "ls linux" or "which linux"

Also search error messages to find meaning or solutions

## Gnu

The Gnu project by the Free Software Foundation programmed most of the commands used within the Linux operating system:

<https://www.gnu.org/manual/manual.en.html>

## RedHat

compile.vcu.edu is running a version of the Linux operating system called RedHat Enterprise Linux version 7

Their documentation is found here:

<https://access.redhat.com/documentation/en/red-hat-enterprise-linux/?version=7>

## Manual pages

man

man <command>

man -k <term> or apropos <term>

This will return the title of all pages can contain the string searched for. Choose your search term wisely. Many of these pages are for the C programming language.

Many GNU commands do not have extensive man pages but use info pages instead.

info / pinfo

Example: compare

man bash

pinfo bash

Look under

Other documentation files can be found in directories under /usr/share/ and /usr/share/doc

Bash examples can be found under /usr/share/doc/bash-\*/examples/

## Safari Books online

Access available through VCU Library. Must be on a VCU network or VPN. Its an amazing research that allows you to search across hundreds of published books.

<http://proquest.safaribooksonline.com/>

## HowTos, Blogs, Wiki

I’ve created some helpful information here:

<https://wiki.vcu.edu/display/unix>

Usually created by users as reminders for projects that they have completed.

<https://wiki.vcu.edu/display/unix>

<https://wiki.vcu.edu/display/vculug>

<https://bitbucket.org/carlisle/wiki/wiki/Home>

It's easy to create a personal wiki by setting up a bitbucket or github account.

# Commands to handle files

## Beginning Commands: ls

The ls command issued by itself will display a list of the names of all the files in the current directory of the shell.

## What are parameters?

The command acts upon files or other system components. The command is usually given first and what the command acts upon is given as addition words after the command. Parameters are separated by delimiters, typically a space. Usually parameters include files that a command opens for reading or writing. Other parameters include switches.

## What is a switch?

A switch changes the default behavior of the command. It allows for different options. They typically start with - or -- Note: there is a space between the ls and -l in the below example:

ls -l

The -l switch is called "long" format and shows additional information such as permissions, owner, group, size, and date of last modification

ls -la

The -a switch stands for "all" and shows hidden files. hiddens files are just like regular files, but they start with a period '.' and are not automatically removed when other files are removed.

ls -larth

Combines many switches together:

"long" format

show "all" files

sort by "time", in "reverse" order so that the most recent files are at the bottom of the list

show sizes in "human" understandable quantities

Commands:

touch

cat

## What are Standard Wildcards?

aka Globbing

Normally used with file names. Not the same as regular expressions.

Standard Wildcards are:

|  |  |
| --- | --- |
| \* | any character, any length, will not match filenames that start with a dot aka hidden files |
| ? | any single character |
| [ ] | set or range of characters, No spaces [a,b,c] or [A-Z] |
| [! ] | negate set or range of characters [!0-9] |
| { term1, term2, term3 } | each term must be a valid wildcard, returns results if any of these are valid |
| \ | (backslash) - escape character |

<http://en.wikipedia.org/wiki/Glob_(programming)>

Compare globbing with regular expressions:

<http://tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm>

Note: These symbols will mean different things when used with regular expressions.

## What is a "Magic Number?"

[http://en.wikipedia.org/wiki/Magic\_number\_(programming)](http://en.wikipedia.org/wiki/Magic_number_%28programming%29)

Command:

file

## History

The bash shell keeps track of your previous commands in a file name .bash\_history. You can use the up and down arrow keys to scroll through the history.

Commands:

cp

mv

cp

cp -ipd

cp -Ripd

mv

Commands:

cat

more

less

head

Tail

tail -n <number> <file>

tail -f <file>

Exercises:

1] Make sure you are in your home directory:

cd

pwd

2] Create a new, empty file

touch testfile

3] List that file showing additional information

ls -l

4] List the files in the /tmp directory

ls -l /tmp

5] List the /tmp directory, sorted by time

ls -larth /tmp

6] Make sure you are in your home directory, copy the directory of bash examples:

cp -Ripd */usr/share/doc/bash-4.2.46/examples/* .

7] Change to that directory and list files

cd examples

ls -l

8] show contents of INDEX.txt file

cat INDEX.txt

9] Also show using more and less

more INDEX.txt

less INDEX.txt

What are permissions?

command:

chmod

ls -ld /usr/share/doc/zip\*

drwxr-xr-x. 2 root root 4096 Jul 14 2015 /usr/share/doc/zip-3.0

file type, read-write-execute for owner, r-w-x for group, r-w-x for others

octal mode

chmod 644

chmod 755

symbol mode

chmod o+r

What is a umask?

Sets the default permissions of newly created files.

umask command with no parameters shows the current umask

umask command with a valid mask sets umask to that mask

Persistent umask changes need to be in ~/.bashrc

umask = 0002 creates files with permissions 664

umask = 0007 creates files with permissions 660

umask = 0077 creates files with permissions 600

# Deleting

rm command

Remember there is no undelete. Once a file is removed, that file is gone (mostly). This is why there are backups.

rm -i - interactive remove. Verification required before removing each file. This is set as the default alias for the rm command on some linux distributions or account, but not on others. It's a good habit to issuing this switch manually.

rm -r - recursive remove. Used to easily remove directories and subdirectories. Use with caution.

Techniques to prevent unwanted removal of files:

First use the ls command with the same switches and wildcards that you want to use with rm, particularly when using wildcards or recursive removes.

**ls -r /home/data/a\***

If the above command doesn’t show any unexpected files, the issue the remove:

**rm -rf /home/data/a\***

========================

# Commands dealing with directories

commands:

cd - change directory

pwd - print working directory

mkdir - make directory

rmdir - remove directory (only works on empty directories)

du - disk usage ( show space used by all files in directory )

<https://en.wikipedia.org/wiki/Unix_filesystem#Conventional_directory_layout>

========================

# Commands dealing with processes

Command:

ps list processes

ps -ef list all processes

ps faux also list all processes as “forest” which processes spawn other processes

What is a pid?

Process Identifier - number in order of when the process was started

pid=1 is always init or systemd

To kill a running process, find its pid and use the kill command.

kill 1290 - this would kill process id 1290

To put a running process into the background, suspend the process using Ctrl-z

A suspended process is still loaded into ram, but not using cpu. To background the suspended process, issue the command: bg

To move a background process into the foreground, issue the command: fg

Commands:

ps

kill

bg

fg

Screen

top

## Pipes and Redirection

What is a Pipe?

|

A pipe allows output from one command to be used as input into a different command.

Example: **fortune | cowsay**

[https://en.wikipedia.org/wiki/Pipeline\_(Unix)](https://en.wikipedia.org/wiki/Pipeline_%28Unix%29)

What is redirection?

>

>>

<

<https://en.wikipedia.org/wiki/Redirection_(computing)>

What is stin, stout, sterr ?

Putting it together:

**ls -l | grep “Jan 31” > search\_results.txt**

**grep “Jan 31” < data.txt > results.txt**

========================

# Commands to use across networks

commands:

ssh Establishes an encrypted session to a remote shell

sftp Transfers files across an encrypted session

scp Also transfer files across an encrypted session, similar to cp

ssh <account>@<server>

sftp <account>@<server>

ls

lls

pwd

lpwd

get

put

scp <file> <account>@<server>: The trailing colon is very important

commands:

wget Used to transfer files from a http, https, or ftp server

curl Also used to transfer files in a slightly different manner

wget <url>

wget -O <newname> <url>

curl will stream url to stdout unless the -O switch is used.

curl -O <url>

curl can be used with the pipe to feed a file into another command

commands:

git version control and get files from a git repository

git clone <url>

========================

# Editing with vim

<https://i.pinimg.com/originals/8f/a8/f2/8fa8f2d4dbfe4810d69e05c945229312.jpg>

The case for vim: There are many editors available on linux, why learn vim?

Its guaranteed to be on any unix system

It uses less memory and bandwidth than gui editors

Many other programs use vi as a built-in editor

It is highly configurable to users’ needs.

## Exiting vim

The challenge to the first-time user of vim is exiting the program. It is unlike most applications. Memorize these keys:

Esc (sometimes more than once), colon, w, q, !, <enter> ( **:wq!** )

Those keys should get you out of any vim session. To explain what they mean:

Esc will get you out of editing mode and into command mode.

**:** (colon) is to issue some commands in vim

**wq!** Issue the commands: write, quit, and do those without verification.

If you are in a split window, sometimes you do <Esc>:qw!<Enter> for every window.

## Command mode

When you start vim you will be in command mode. Nothing on the screen will indicate this.

Some commands require a colon ( **:** ) followed by enter and some do not. There is a difference between the command, **x** (delete a character), and the command **:x<Enter>** (exit, like :wq, but only writes if changes have been made)

Common command used are:

**:w** Write the current file

**:q!** Quit the current file, without saving it, don’t verify

**:wq!** Write and quit the current file, don’t verify.

**dd** Delete the entire line where the cursor is located.

**:help** Show built-in help system :help <command> will jump to the help section for that command. Try **:help :wq** To exit help, use **:q**

## Editing mode

There are many commands that can be issued to enter into editing mode, the most common are:

**i** For insert. This will allow you to start typing at the current location of the cursor.

**a** For append. This will allow you to start typing at one position to the right of the cursor. Best for adding new content at the end of the line.

**o** Open a new line and start editing

**x** Delete the character under the cursor

**u** Undo the last edit

To exit editing mode, press the Esc key. You will then return to command mode.

<https://stackoverflow.com/questions/14051712/why-some-commands-in-vim-require-a-colon-while-some-dont>

## Vimtutor

The easiest way to learn how to use vim is this built-in tutoring system. It will pull up vim with a prewritten file that has exercises on how to use vim. Any changes to the file are erased when you exit. If you make a mistake, exit, and you can start again with a fresh file.

**vimtutor**

<http://wall-skills.com/2014/vim-cheat-sheet/>

<https://www.nostarch.com/mug.htm>

## Advanced commands

search and replace

**:substitute** abbreviated **:s**

**:%s/foo/bar/g**

If search and replace is needed for strings with slashes within them, you can change the delimiter to another character. # is used in this example: search for all strings “/etc/foo” and replace them with the string “/etc/bar”

**:%s#/etc/foo#/etc/bar#g**

<http://vim.wikia.com/wiki/Search_and_replace>

Turn off auto indent

**:set paste**

<https://stackoverflow.com/questions/2514445/turning-off-auto-indent-when-pasting-text-into-vim>

========================

# Commands for handling strings: searching, cutting, replacing

command:

find Used to find a file with the given name or attributes

command:

grep Used to find a string within a file

command:

locate Quickly find a file name from a simple database of indexed files, updated nightly.

## find

The **find** command searches for a filename of the given pattern within a given directory

**find <directory> -name <pattern> -print**

<pattern> uses globbing

**-type f**  return only files

**-type d** return only directories

**find / -name file -type f -print**

**find / -regex <regex>**

While find can be used to pipe output to another command, some commands can’t accept this so the **-exec** switch is used instead. **{}** is the output from find and it must end with **\;**

**find . -name "\*.txt" -exec echo {} \;**

**xargs** - many commands have a maximum number of arguments they can process, the xargs command breaks down an arbitrarily long input stream into chunks of arguments that can be processed by another command

<https://en.wikipedia.org/wiki/Xargs>

**-print0** ignores spaces as delimiters allowing the processing of file names with spaces such as you find on Windows or OSX.

Putting these together with find will result in this handy template for dealing with a large number of files, possibly with spaces in the name.

**find <directory> -name <file> -print0 | xargs -0 | <command>**

## grep

The **grep** command searches for strings within a file. Upon a successful match, the line that contains the string is displayed.

grep <expression> <file>

grep -i = case insensitive

grep -v = exclude

grep -x = exact match

expression can be expressed using globbing or regex

<https://www.gnu.org/software/grep/manual/grep.html>

## Regular Expressions

<https://en.wikipedia.org/wiki/Regular_expression>

They expand upon wildcards and allow for very complex searches. Regular expressions may have slightly different forms in different programs. The most common is perl-compatible regular expressions (pcre)

Basic regular expressions:

**^<string>** - look for string only at the start of a line

**<string>$** - for for string only at the end of a line

**.** - the dot or period represents match any character, similar to \* in globing

**\*** - the star or asterix represents repeat of previous character zero or more times

<https://regex101.com/> Online regex tester and debugger

<http://www.regular-expressions.info/>

Examples:

Show all lines of a file that begin with # ( most likely comments )

**grep "^#" <file>**

Show all lines of a file that don’t begin with a # or ;

**grep -v "^#" <file> | grep -v “^;”**

**grep "^[^#;]" <file>**

<https://unix.stackexchange.com/questions/60994/how-to-grep-lines-which-does-not-begin-with-or>

## sed awk cut

Printing the nth column of text data

With cut: (fixed delimiters)

echo "abc 123 ABC 123" | cut -f 3 -d ' '

With awk: (arbitrary amount of whitespace)

echo "abc 123 ABC 123" | awk '{ print $3 }'

<https://unix.stackexchange.com/questions/132313/what-are-the-exact-differences-between-awk-and-cut-with-grep>

<https://stackoverflow.com/questions/1632113/what-is-the-difference-between-sed-and-awk>

<http://mindspill.net/computing/linux-notes/text-manipulation-cut-awk-and-sed/>

<https://stackoverflow.com/questions/17813244/string-manipulation-with-awk-sed-and-cut-together>

<https://linuxcommando.blogspot.com/2008/03/using-sed-to-extract-lines-in-text-file.html>

# File archives, compression, and checksums

## tar

Tar originally stood for tape archive. Most users no longer use it to backup to tape, but as a way to combine many files and directories into one file for easy transfer.

Extract a tar archive: pull all files and subdirectories within file.tar into current directory

**tar -xvf file.tar**

Create a tar archive: puts all files and subdirectories of /directory into archive named file.tar

**tar -cvf file.tar /directory**

## Compression

Each of these are compression methods to reduce the size of files without the loss of information

**gzip**

**gunzip**

**bzip2**

**bunzip2**

**xz**

Since compression is frequently used with tar, the tar command is aware of these compression methods and can be called from within tar. For example the following will uncompress with gunzip and then extract a tar.gz file

**tar -xvzf file.tar.gz**

**tar -xvjf file.tar.bz2** for bzip2 compression

**tar -xvJf file.tar.xz** for xz compression

Finally the most recent version of tar will detect the compression method used and don’t need to be stated explicitly. These should all work on recent linux distributions:

**tar -xvf file.tar.gz**

**tar -xvf file.tar.bz2  
tar -xvf file.tar.xz**

<https://www.howtogeek.com/248780/how-to-compress-and-extract-files-using-the-tar-command-on-linux/>

## checksums

These use hash functions used to generate and verify files. Checksums are important to verify that a file has not been altered since the checksum file was created.

**md5sum**

**sha1sum**

**sha256sum**

**sha512sum**

md5sum --check file.md5 will look at a file of md5 checksums named file.md5 and verify the checksum for each of files listed.

## Md5deep

## Signatures

<https://www.apache.org/info/verification.html>

<https://stackoverflow.com/questions/15331015/how-to-verify-downloaded-file-with-sig-file>

<https://www.cyberciti.biz/faq/pgp-tarball-file-signature-keys-verification/>

# Bash Scripting

Puts bash commands into a file and allows you to run a complex series of commands over and over.

## The Shebang

**#!** - is an interpreter directive

Scripts are just text files. In Windows and other operating systems the file extension ( .sh .py .pl etc ) lets the operating system know how to handle the file, but not in linux. One must set the Sheband correctly.

The Sheband tells the operating system which interpreted to run the script file through. Must include the absolute path to the interpreter.

Examples:

Bash #!/usr/bin/bash

Python #!/usr/bin/python

Perl #!/usr/bin/perl

It can include parameters

#!/usr/bin/perl -T

Execute using Perl with the option for taint checks

This can also be made more portability across different unix and linux operating systems with the env command. This looks up the absolute path on a given operating system.

#!/usr/bin/env bash

[https://en.wikipedia.org/wiki/Shebang\_(Unix)](https://en.wikipedia.org/wiki/Shebang_%28Unix%29)

Its best to give scripts executable permissions so that they will run when their name is used

## A simple bash script

Create a script called hello.sh with the following:

**#!/usr/bin/env bash**

**echo "Hello World!"**

Try to run it:

**./hello.sh**

Now try to run it with:

**bash ./hello.sh**

Now give it executable permissions:

**chmod 700 hello.sh**

or

**chmod u+x hello.sh**

Now try to run it.

## Bash variables

Bash scripts frequently use variables to simply scripts and make it easier to change.

[ show example ]

Variables can also be the output of commands:

Create a file called date.sh with the following:

**#!/bin/env bash**

**DATE=$(date +%F)**

**echo $DATE**

In older scripts it may use backticks for the the same purpose. This is now deprecated

**DATE=`date +%F`**

## echo vs print

[add more here]

## Special Variables: $0 $1 $2 $#

$0 is the script itself

$1 is the first argument to the script

$2 is the 2nd argument

et cetera

$# is the number of arguments passed.

Create this script: arguments.sh

**#!/bin/env bash**

**echo "Script is: $0"**

**echo "Number of arguments are: $#"**

**echo "First argument is: $1"**

**echo "2nd argument is: $2"**

./arguments.sh

./arguments.sh Hello

./arguments.sh Hello World

<http://tldp.org/LDP/abs/html/othertypesv.html>

Conditionals:

If-then structures in bash take the form of:

**if [ <condition> ]**

**then**

**<something>**

**else**

**<something\_else>**

**fi**

Notice where the line breaks are. The above can also be written as:

**if [ <condition> ]; then**

**<something>**

**else**

**<something\_else>**

**fi**

OR

**if [ <condition> ]; then <something>; else <something\_else>; fi**

Typical conditions:

**T1="foo"**

**T2="bar"**

**if [ "$T1" = "$T2" ]; then**

**echo expression evaluated as true**

**else**

**echo expression evaluated as false**

**fi**

Note there is a space before and after =

<http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-6.html>

AND

**if [ -n $1 ] && [ -r $1 ]**

**then**

**echo "File exists and is readable"**

**fi**

<http://bencane.com/2014/01/27/8-examples-of-bash-if-statements-to-get-you-started/>

Conditionals also can be used to do certain file tests:

One of the most useful is to test if a file exists:

Example 1:

**if [ -e <file> ]; then <something>; fi**

Example 2:

**FILE=~/.bashrc**

**if [ -e $FILE ]; then tail $FILE; fi**

Example 3:

**if [ ! e $FILE ]; then exit; fi**

Test if a file is a regular file ( and not a directory or other file type )

**if [ -f $FILE ]; then tail $FILE; fi**

A file is of non-zero size:

**if [ -s $FILE ]; then tail $FILE; fi**

<http://www.tldp.org/LDP/abs/html/fto.html>

Note:

There are also double brackets for conditions [[ <condition> ]] this is called the extended test command

There is also the double parentheses (( <condition )) which expands an arithmetic expression.

<http://tldp.org/LDP/abs/html/testconstructs.html#DBLBRACKETS>

Some special properties of loops:

Creating a loop with sequential iteration:

**for i in {1..25}; do echo $i; done**

also letters

**for i in {a..z}; do echo $i; done**

<http://www.cyberciti.biz/faq/bash-for-loop/>

Bash allows you to process all files in a directory through a loop:

**#!/bin/env bash**

**FILES=*/path/to/*\***

**for f in $FILES**

**do**

**echo "Processing $f file..."**

**# take action on each file. $f store current file name**

**cat $f**

**done**

<http://www.cyberciti.biz/faq/bash-loop-over-file/>

Exception Handling in bash: <https://stackoverflow.com/questions/22009364/is-there-a-try-catch-command-in-bash>

## Where to find more bash help

Bash Programming HOWTO

<http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>

Bash Guide for Beginners

<http://www.tldp.org/LDP/Bash-Beginners-Guide/html/>

Lhunath's Bash Guide

<http://mywiki.wooledge.org/BashGuide>

Advanced bash scripting guide

<http://tldp.org/LDP/abs/html/>

Goalkicker Bash Book

<http://goalkicker.com/BashBook/>

Note: bash has been around since 1989 and has changed over the years. When searching for help online, look for the latest solution to a problem.

========================

# Batch processing with gridengine

Gridengine is a scheduler and resource manager. It has gone by many names but is essentially the same program: SGE, Sun Grid Engine (SGE), Oracle Grid Engine, Univa Grid Engine, Open Grid Scheduler, and Son of Grid Engine (also SGE)

Batch processing, also known as job scheduling, batch scheduling

Other High-Performance Cluster (HPC) batch systems are: PBS, Condor, SLURM, Moab

Basic idea is that it allows the user to put one or more jobs into the background and the operating system will determine when the job is run based on resources available.

Unfortunately, the documentation for gridengine is arcane and hard to read.

Other notes are here:

<https://wiki.vcu.edu/display/unix/GridEngine>

It is best to create a wrapper script for the application to be submitted to gridengine. The wrapper script is a bash script that contains all the gridengine parameters plus and environmental variables needed to run the application.

The wrapper script is a bash script which includes all environmental variables needed to run the commands. The user environment is not inherited by gridengine, so if you set up a custom path or other variables in your user shell, you need to do the same within the wrapper script.

The wrapper script also contains parameters to the qsub command itself. Which start with #$

The equivalent of the command:

**qsub -cwd -e ./ -o ./ -S /bin/bash -N environment <script>**

is the same as the following within the script:

**#!/usr/bin/env bash**

**#$ -cwd**

**#$ -e ./**

**#$ -o ./**

**#$ -S /bin/bash**

**#$ -N environment**

**<script>**

========================

# Batch Processing with slurm

<https://slurm.schedmd.com/pdfs/summary.pdf>

<https://secure.hosting.vt.edu/www.arc.vt.edu/slurm-user-guide/>

<https://support.ceci-hpc.be/doc/_contents/QuickStart/SubmittingJobs/SlurmTutorial.html>

gridengine equivalent commands:

sinfo

qstat -> squeue

qsub -> sbatch

qrsh -> use the following two commands to request a resource and then use that resource.

salloc -N 1 -t 10:00 ( request a resource, one slot, for 10 minutes )

srun --pty bash ( get a bash shell on requested resource

qdel -> scancel <jobid>

====

# Compiling Basics

Create simple C program named hello.c

**#include <stdio.h>**

**int main (void)**

**{**

**printf ("Hello, world!\n");**

**return 0;**

**}**

Compile hello.c using the command:

**gcc -Wall hello.c -o hello\_c**

<http://www.network-theory.co.uk/docs/gccintro/gccintro_9.html>

Run it, using the command:

**./hello\_c**

Create simple fortran program named hello.f90

**program hello**

**print \*, "Hello World!"**

**end program hello**

Compile it using the command:

**gfortran -Wall hello.f90 -o hello\_f90**

Run it use the command:

**./hello\_f90**

Mixed programming example:

calling fortran function from c program. [work on this]

<https://www.hoffman2.idre.ucla.edu/c-fortran-interop/>

<https://computing.llnl.gov/tutorials/bgq/mixedProgramming2.pdf>

## GNU Build system

Many open source software projects use the GNU Build system

The usual procedure is to cd to the code directory and do:

./configure --prefix=<target\_directory>

make

make install

<https://en.wikipedia.org/wiki/GNU_Build_System>

autoconf

autoconf creates the configure script. Not all projects will need autoconf run by users.

<https://en.wikipedia.org/wiki/Autoconf>

configure

<http://en.wikipedia.org/wiki/Configure_script>

configure creates the Makefile that is appropriate for your system. The default target directory of the configure script is /usr/local but only administrators have access to that directory, so if you are not an administrator, you will need to designate a new directory that you have privileges to write to.

--prefix=<directory>

[http://en.wikipedia.org/wiki/Make\_(software)](http://en.wikipedia.org/wiki/Make_%28software%29)

make runs the Makefile which includes all the compiler instructions and then link multiple object files if they are created.

make install will take the binaries created by the make command and put them into the target directory

There are other build systems but this as the most widely deployed.

========================

# Putting it all together: A compiling example:

1. Create a directory called "src" under your home directory

2. Change to that directory and download a file using either method A or method B

Method A,

Step 1: Select one of these urls to download the source code:

<http://rogue.rogueforge.net/files/rogue5.4/rogue5.4.4-src.tar.gz>

<http://pkgs.fedoraproject.org/repo/pkgs/rogue/rogue5.4.4-src.tar.gz/033288f46444b06814c81ea69d96e075/rogue5.4.4-src.tar.gz>

<http://mirror.vcu.edu/vcu/stuff/rogue5.4.4-src.tar.gz>

Also download this checksum file:

<http://mirror.vcu.edu/vcu/stuff/rogue5.4.4-src.tar.gz.md5sum>

Step 2: Verify the checksum

Step 3: Extract the contents from rogue5.4.4-src.tar.gz

Method B

use git:

**git clone****<https://github.com/phs/rogue>**

**mv rogue rogue5.4.4**

3. Change to the rogue5.4.4 directory

4. Run configure and set the target of installation to be the directory "rogue" under your home directory /home/<account>/rogue5.4.4

6. Make and install the code using the steps of the GNU Build System

7. Determine if the rogue program is in your path

8. Correct an error in how the man pages were installed:

change to $HOME/rogue5.4.4/share/man

make the directory "man6"

move the file rogue.6 to the directory man6

9. Create a file with all the environmental variables needed to run this program at $HOME/rogue5.4.4/rogue\_env.sh Append the PATH to include the rogue binary and also export the MANPATH varible and set it to $HOME/rogue5.4.4/share/man

10. source the file created above

11. run rogue

12. Enjoy

A step-by-step set of instructions for completing this task can be found in the directory compile.vcu.edu:/usr/local/

What is rogue?  
  
<http://www.roguebasin.com/index.php?title=Rogue>

<https://en.wikipedia.org/wiki/Rogue_(video_game)>

“Among its fans included UNIX's co-developer Ken Thompson working at [Bell Labs](https://en.wikipedia.org/wiki/Bell_Labs); Dennis Ritchie had joked at the time that *Rogue* was "the biggest waste of CPU cycles in history"

## Other early computer games: Adventure

<http://www.amc.com/shows/halt-and-catch-fire/exclusives/colossal-cave-adventure>

Suggestions to get started:

Look, Go building, Get lamp, Get keys, Get food, Get water, Go outside, Follow stream

Go south

If you want to know more about these type of games search for the documentary: Get Lamp <http://www.getlamp.com/>

The original source code is difficult to install on modern operating systems. A modern effort to recreate the original game is here:

<https://gitlab.com/esr/open-adventure>

====

Future note:

Elements of the Linux or Unix operating system

Kernel

Systemd ( or init on older systems )

System logger

Cron scheduler

Networking stack

Shell

User commands: Gnu