

Course notes for:

Computational Tools for Geoscientists (GEOL 4002)

Fall 2008

LSU

Baton Rouge

Juan M. Lorenzo

Table of Contents

[Acknowledgements 4](#_Toc211151881)

[Introduction 4](#_Toc211151882)

[Why do we need to study linux? 4](#_Toc211151883)

[Why do we need OpenSource software? 4](#_Toc211151884)

[Where do I get ssh? 4](#_Toc211151885)

[Are are planning on doing any programming from home? 4](#_Toc211151886)

[Where do I get Xming? 4](#_Toc211151887)

[How to run Xming: 4](#_Toc211151888)

[Why do we need to know sh or Perl? 5](#_Toc211151889)

[Linux 6](#_Toc211151890)

[Linux 6](#_Toc211151891)

[History of Linux 6](#_Toc211151892)

[Linux Shells (Albing et al., 2007) 6](#_Toc211151893)

[Directory Structure of the Linux operating system 7](#_Toc211151894)

[Additional useful linux instructions 10](#_Toc211151895)

[Vi (Visual Editor) 20](#_Toc211151896)

[Perl {Hoffman, 2001 #3408} “**P**ractical **E**xtraction and **R**eport **L**anguage” 23](#_Toc211151897)

[Why use Perl? 23](#_Toc211151898)

[When not to use Perl? 23](#_Toc211151899)

[Tutorials 23](#_Toc211151900)

[Basic components of Perl 24](#_Toc211151901)

[Exercises: 33](#_Toc211151902)

[Directory structure and file locations 33](#_Toc211151903)

[Do-loop 33](#_Toc211151904)

[Awk 33](#_Toc211151905)

[Perl lists 34](#_Toc211151906)

[Write out Lincoln’s Gettysburg address using lists. 34](#_Toc211151907)

[Perl for-loop 34](#_Toc211151908)

[Perl write to a file 34](#_Toc211151909)

[Perl read from a file 34](#_Toc211151910)

[Perl **if** logical operator 34](#_Toc211151911)

[References 37](#_Toc211151912)

## Acknowledgements

Many students have contributed to these notes:

Class of 2008: Erin Walden, Kody Kramer, Erin Elliott, Andrew Harrison, Andrew Sampson, Ana Felix, JohnD’Aquin, Russell Crouch, Michael Massengale, David Smolkin

## Introduction

### Why do we need to study linux?

Creative professional geophysicist and academics are able to explore new ideas without constraints of “black-box” software.

### Why do we need OpenSource software?

Scientifically, open source products can be verified independently by anyone. Reproducibility is a core tenet of the scientific method. OpenSource software replicates a scientific procedure.

### Where do I get ssh?

Link to ssh: [*http://web.wm.edu/it/?&id=2948&svr=www*](http://web.wm.edu/it/?&id=2948&svr=www)

### Are are planning on doing any programming from home?

* Open SSH. Create a profile named ‘odyssey’.
* Now go to File> Profile>Edit Profile. Edit the ‘odyssey’ profile.
* On the Connections tab: Hostname field: odyssey.geol.lsu.edu (IP 130.29.168.63) Username: the user name given you in class by Dr. Lorenzo. Your password is of the form XXXXXXX, where XX is a number given you by Dr. Lorenzo. The password is case-sensitive. Save changes to your profile.
* You can now connect to the odyssey server using SSH.

### Where do I get Xming?

Xming is the leading, free X Windows Server for Microsoft Windows.

For notes link to Xming: <http://www.straightrunning.com/XmingNotes/>

For download of X fonts, use Google, e.g: <http://sourceforge.net/projects/xming>

For download of Xmin server, <http://download.cnet.com/Xming/3000-2094_4-10549058.html>

### How to run Xming:

* Making sure that you’re still connected in SSH, run XLaunch to configure Xming to connect to odyssey. Choose one window, then make sure that “Start no client” is checked. Click Next>Next>Finish. Logout of SSH(File>Disconnect) and then reconnect by selecting the odyssey profile.
* If you are having problems connecting, open the odyssey profile in SSH and go to Edit>Settings. Under the Tunneling option on the tree, make sure that the “Tunnel X11 Settings” option is checked. Make sure to save your profile.
* You now know you correctly edited the .login file if it reads DISPLAY: undefined variable. If you get something with error in it, check to make sure the setenv line is commented out.

### Why do we need to know sh or Perl?

Shells are the basic sets of instructions for handling the operating system and perl is a mature, widespread computer language ideal for file manipulation. Perl can serve as a simple “glue” to make diverse pieces of software talk to each other.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Purpose | Type | Niche | Easiest OS |
| sh | command language interpreter , i.e., OS instructions | Low-level | Program the OS | Linux |
| [Perl](http://www.perl.org/) | Scripting language with tools like in C or Fortran | Low-level text-based | "Glue" for all other programs | Linux, MacX Windows |
| [Matlab](http://www.mathworks.com/) | computational programming | High-level | Matrix manipulation | Linux, MacX,Windows |
| [GMT](http://gmt.soest.hawaii.edu/) (Generic Mapping Tools) | Quantitative analysis and display of 2D,geographically referenced geophysical data sets | Low-level C programs | Marine geophysics | Linux, MacX (Windows native or under \*Cygwin) |
| [Strata](http://hydro.geosc.psu.edu/Strata_html/man.html) | Interactive 2D modeling of basin stratigraphy | Interactive | Sedimentary analysis of basins | linux |
| [GRASS](http://grass.itc.it/) | Interrrogation, DB, calculations and displays of 2D, 3D vector-based geographic data sets | Low-level C programs | Surface Process | linux |
| MBSys | Quantitative analysis and display of 2D,geographically referenced sonar data sets | Low-level C programs | Marine geology | linux |

### Linux

The single-most advantage of linux is that the code is freely available so many people around the world participate in its improvement continuously. I first view Linux as a communal, philanthropic exercise which takes advantage of the cooperative nature of our species. Linux is also a collection of instructions in software that allow you to use the hardware in your computer.

**If** well thought out, visually identifiable commands are friendlier if but slower to use, (although especially tedious to write and computationally less efficient). As part of linux there is a “point and click” WYSIWYG (“What-you-see-is-what-you-get”)/GUI(“Graphical-user-interface”) to drive the same instructions, visually.

## Linux

### History of Linux

Click [here](https://netfiles.uiuc.edu/rhasan/linux/#Table%20of%20Contents) for a more comprehensive history of the subject by Ragib Hasan at **UIUC.**

Linux was developed (for free) by Linus Torvald possibly inspired by at least the [GNU project](http://www.gnu.org) (“GNU’s not Unix”) , a software movement to provide free and quality software.

LINKS to sites that have important shell instructions:

[Important Instructions in sh](http://www.geol.lsu.edu/jlorenzo/linux_commands/linux_commands.htm)

### Linux Shells (Albing et al., 2007)

#### Q. What is a shell?

A shell is a convenient collection of command-line-instructions (actual programs), written in a low-level language, such as C, which allow the user to interact with files and the hardware and files. Shells have been around since the start of the unix-type operating systems and have the advantage that they interchangeable among different linux operating systems. Although the instructions may have to be recompiled for each machine the syntax remains constant and once learnt will last a career.

Example, *ls*.

*ls* stands for: “list the contents of this directory”

#### Q. Why are there different shells?

#### Q. What are the different shells?

##### sh: the original “bourne-shell”

##### csh: the“C-shell”

The csh improves upon the sh because it introduces convenient programming tools inherited from C

##### ksh: the “k-shell”

The commercial nature of this shell limited the growth of its popularity from the start.

##### bash: the “bourne-again-shell”

The bash shell is ubiquitous among any linux-type operating systems you might encounter. The bash shell inherits the advantages and experiences of all prior shells.

#### Q. Which one should I use?

For this class the default is: csh

### Directory Structure of the Linux operating system

In any operating system, linux programs and user directories are stored in predictable locations. [Exercise](#Exercise_linux_drectory_strct)

Q. Do you know where the passwords are kept? [Exercise](#Exercise_linux_password)

Q. What are “system permissions”?

Every file and directory in linux has assigned codes which dictate the degree of authority by each user of the computer to alter each file. There are four types of user status on linux. First is the overall supreme administrator known as “root” and who can do anything to any file on the system. Next comes the specific original owner/user of each file. All users can belong to one or several named “groups” of users. Finally anyone who is not specified as belonging to your group or is not the supreme adiministrator is considered belong to the outside “world”, or all other users. Within each of the status levels: owner, group, world, binary codes or their letter equivalents may be set to indicate whether a file may be only browsed (“read”), modified (“write”), and/or executed as a program (“executable”). Note that it is the files themselves that carry this important information with them. The file permissions are consulted first to determine whether an individual user has authority to manipulate the file in any way.

The purpose of this complex permission scheme is to provide an infinite variety of protection schemes for the file systems but yet maintain an unsinkable file system. In theory, and for much of practice, an individual user will not be able to shut down the system; they will only be able to do damage to themselves and not the files or others.

System permissions belonging to a file or directory can only be changed by those users for whom files have had the proper permissions already assigned. Initially it is “root” that sets all the first set of permissions for files and directories when a user is given a space to work on the system. From the first logon, the user has control of their assigned set of files and directories.

If you want a file containing Perl code to become executable in the system the creator of the file is required to change the appropriate permission setting for that file. Following are the equivalent numeric codes for the different types of permissions:

Read only - 4 Write only – 2 Execute only - 1

Read and write – 6 Write and execute – 3 Read, write and execute – 7 (add all three numbers together)

For example:

% ls –l

My\_perl\_file r \_ \_ r \_ \_ r \_ \_

There are three spaces to explain the type access by user:

(“read” access), group (nothing/0) and everyone-else (dash/0), respectively. The next three spaces show the same for the group to which the user belongs and the final three for all other users.

In order to change “permissions” to allow the file to run as a program enter the following:

chmod u+x

which only adds (“+”) the setting that gives only the owner (“u”) executing privileges

Or, equivalently

chmod 600

In the numeric form the last two zeros mean that “group” and “others” priviliges are nill. As you can see the numeric form can alter permissions for all the three types of linux users at once.

Here is a summary list of options used for setting file permissions and understanding file types on the linux system

|  |  |  |  |
| --- | --- | --- | --- |
| Abbreviation of user status | Stands for … | Abbreviation of file permission | Stands for … |
| u | user | r | read |
| g | group | w | write |
| o | others | x | execute |
| a | all |  |  |
| + | add |  |  |
| - | remove |  |  |
| d | directory |  |  |
| l | link |  |  |

Examples:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Letters symbols | Numerical symbols |  |  |  |
| chmod u+rwx | chmod 700 |  |  |  |
| chmod u+rwx  chmod g+rw  chmod o+x | chmod 761 |  |  |  |

Q. Can I do any damage to another person’s files?

Yes, if the files belong to you. You can tell if you own the files by reading the second column from the ls –l instruction, which has the general form

drwxr-xr-x “number of links” “your login name” “your group name” filesize(bytes) date etc.

### Additional useful linux instructions

#### System Instructions

##### [Moving](#cd) [Around](#cd)

##### [Logging In](#logging)

##### Review previous instructions

##### [Running a Remote Session](#remotesession)

##### [Running a program](#runningaporogram)

##### Help manuals

##### [Secure file copying across the internet](#sftp)

#### Moving Around

If you are lost in the system and need to get back to your own directory, an alias (shortcut) has been generated for you in a hidden environment script:

% cd

If you want to relocate yourself in the system, e.g., go to the directory that contains the passwords:

% cd /etc

[TOC](#TOC) [System](#systeminstructions) [Instructions](#systeminstructions)

#### Logging In

Type your login id, followed by your password

[TOC](#TOC) [System](#systeminstructions) [Instructions](#systeminstructions)

#### Review previous instructions

Currently, up to about 60 of the latest comman-line-instructions you have entered are stored while you work in linux. If you want to see what they are input:

%history

You will immediately get a list of all the instructions you have recently entered and each successive instruction is identified by a number that appears first on each line. If you want to repeat any particular instruction enter an exclamation mark followed by the instruction number:

% !instruction\_number

[TOC](#TOC) [System](#systeminstructions) [Instructions](#systeminstructions)

#### Running a remote session

ssh [yourname@odyssey.geol.lsu.edu](mailto:yourname@odyssey.geol.lsu.edu)

setenv DISPLAY *localhost:10.0* (redirect images to the machine you are sitting at)



Answer "yes" to the question involving "authenticity". You may only see this question the first time you log on from each machine.

You should see a "prompt" such as

%odyssey:/home/yourname %

To see what is in your directory:

%ls –l

To see everything in your directory, even hidden files (.\*):

%ls –la

[TOC](#TOC) [System](#systeminstructions) [Instructions](#systeminstructions)

#### Running a program

In order for a file to become a program, it must be executable.

#### 

[TOC](#TOC) [System](#systeminstructions) [Instructions](#systeminstructions)

#### Help Manuals

Online manuals for each shell instructions can be called via the “man” command, e.g.,

% man cd

% man ls

% man pwd

Once you are in a help manual you can move around inside by using keyboard shortcuts which are listed within each manual. If you want to make a short help list appear, type “h”. In order to find specific text within a manual, input

“/a\_specific\_word”

For example, the following instruction entered from within the manual page for “ls” looks for the first occurrence of the switch “-l”

/-l

[TOC](#TOC) [System](#systeminstructions) [Instructions](#systeminstructions)

#### Secure file copying across the internet

#### Using SSH-secure FTP

##### Double-click on the ssh file transfer icon

##### When prompted, enter your password

##### Click connect and Enter

##### To transfer the file, just drag and drop into the desired directory

##### Another way to do this is to set up a program that will do it for you

##### At the prompt: odyssey:/username% enter sftp [email@domain.com](mailto:email@domain.com) or machine name (e.g. odyssey.geol.lsu.edu)

##### They will then exchange information and ask for a password.

##### You can then copy from your local account to wherever you like.

##### But for our purposes, drag and drop is sufficient.

##### The ssh file transfer allows you to see the file transfer pane and the local directory at the same time.

##### With SFTP you have to connect and interact with another server

#### If you are using a linux box or a Macintosh (with MacOSX)

##### At the prompt: odyssey:/username% enter

^% sftp [email@domain.com](mailto:email@domain.com) or

% machine name (e.g. odyssey.geol.lsu.edu)

##### They will then exchange information and ask for a password.

##### You can then copy from your local account to wherever you like.

Once you are connected to the remote machine the following basic instructions will get you going:

help

get download a file over to the directory on the local machine

put upload a file to the remote machine

ls list CONTENTS of the remote machine

lls list directory CONTENTS of the local machine

pwd l working directory name of the remote machine

lpwd working directory name of the local machine

history gives a list of recent instructions

!”number” repeats instruction “number” from history list

!”Letter” repeats last instruction in history list that starts with “Letter”

Tab automatic completion of a file name

^D When file completion does not complete the file it means that another file name may exist with the same beginning letters. By entering a “control-D” you can inspect the names of these other ambiguous files. Continue within file completion by entering the next character.

[TOC](#TOC) [System](#systeminstructions) [Instructions](#systeminstructions)

#### File Manipulation

##### [Archive of directories and their contents](#Tarring)

##### [Assigning values to variables in shell scripts](#Assigning_values2variables_shells)

##### [Creating a shell script to log in automatically](#script2login)

##### [Comparing files](#comparingfiles)

##### [Concatenating files](#concatenatefiles)

##### [Copying a file to your home directory](#copyafiletohomedirectory)

##### [Deleting Files](#deletingfiles)

##### [Duplicating files](#duplicatingfiles)

##### [Editing a file](#Editing_a_file)

##### [Examining content without directly opening the file](#greppingfiles)

##### [Finding files](#findingfiles)

##### [Renaming files](#renamingfiles)

##### [Repetitive tasks](#repetitivetasks)

##### [Screen dump](#screendump)

##### Changing file content- awk

#### Archive of directories and their contents

When it comes to collating all your directories and their contents into a single manageable file that can keep a record of the directory structure use the instruction called *tar* as follows:

%*tar –cvf tarred\_file\_name* *directory\_to\_archive*

A file called *tarred\_file\_name* is created. Usually it is best to give your tarred file a \*.tar ending so you can automatically know what type of file it is in future. In order to open up and generate all directory tree with all its leaves (which are the files contained within ) use the following command:

*% tar –xvf tarred\_file\_name*

If you choose to get ONLY a LISTING of the contents of a tarred file without rebuilding the directory tree and all its contents you can instead use the following command:

*% tar –tvf tarred\_file\_name >output\_file* or if you want to output the listing to the screen use:

*% tar –tvf tarred\_file\_name*

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### [Assigning values to variables in shell scripts](#Assigning_values2variables_shells)

A free linux shell scripting tutorial:

<http://www.freeos.com/guides/lsst/>

|  |  |
| --- | --- |
| Example 1  The text ‘hello’ is assigned to the variable named *output* The value of the variable is expressed as $*output* The variable name can be any word. | Example 2  The number *1* is assigned to the variable named *value.* The value of the variable is expressed as $*value*  $*1* is assigned value 2 from the command line (outside the shell script). This number *is* the first value on the command line after the *prog name*  Arithmetic calculations are carried out by a shell program called *expr.* |
| %prog\_name | %prog\_name 2 |
| #!/bin/sh  Output ’Hello, handsome’  echo $output | #!/bin/sh  value=1  $new\_value = $[1+$value]  echo $new\_value |

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### [Creating a shell script to log in automatically](#script2login)

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### Comparing files

%diff file1 file2

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### [Concatenating files](#Concatenatingfiles)

When you have one files you would like to append to another use the

%cat file1 file2 > file3

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### Copying a file to your home directory

To be safe, don’t work on the original version of a file. Make a copy

Type in cp (which means copy) and then the file name:

% cp (FROM)filename (TO)filename2

If you don’t remember the file name, hit the tab key and it will try to find the file name for you.

Example: Copy test 2.sh

Type:

% cp test2.sh test1.sh

You have now made a copy of test2.sh named test1.sh

To test the copy

Type

%ls –l

A list of programs will pop up and you should see test1.sh listed.

#### 

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### [Deleting Files](#Deleting_files)

**%**rm filename

In order to delete a file without any bothersome questions from the operating system enter:

%rm –rf filename

The following instruction will mean INSTANT DEATH and RIDICULE In order to delete everything you own system enter:

%rm –rf \*

**Instant death** for your grade and will result in many people laughing at you

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### Duplicating files

%cp file1 file2

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### [Editing a file](#editfile)

% gedit .cshrc

Move down (use cursor arrows) until you see the line that has the word "source" in it and on the next line enter:

% source /usr/local/admin/cshrc\_local

Exit gedit and enter the following instructions:

% source .cshrc (updates the behavior of your operating system)

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### Examining content without directly opening the file

**%** grepsearchtext filename(s)

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### [Finding files](#Finding_files)

**%** locatefilename

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### Finding files in combination with grep

This instruction can be very useful when you are trying to locate a file from among many others.

% slocate perl | grep bin | grep usr

In the example above we are attempting to locate “perl”

We know that “perl” exists within a directory called “bin”

If we do not use the grep command and only the first “slocate perl” instruction the output to the terminal will be long and confusing. To find this out for yourself try the following alone:

% slocate perl

Once you see this output you can try the following, slightly longer sequence:

% slocate perl | grep bin

The second “grep bin” commands filters out the preceding stream of text to those pieces that have “bin” in the file name or its complete directory address

Finally try the full instruction:

% slocate perl | grep bin | grep usr

You should have notice that with each additional “grep” command the final output list becomes smaller because we are restricting the search for additional text items.

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### [Renaming files](#Renaming_files)

% mv old\_file\_name new\_file\_name

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### Repetitive tasks

#/bin/sh

**for** action in 'came. ' 'saw.' 'conquered.'

**do**

echo 'I ' $action

**done**

The variable called *action* can have three values. Each value is a word that is sent to the screen using echo within the **do** …**done** set of instructions. The $ sign in front of *action* assigns its value to be sent to the screen each time following the word *I*. [Exercise](#Do_loop)

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### Screen dump

**%** gimp

Experiment by dumping an image of the screen into a file, opening it and then printing it.

-GIMP (GNU Image Manipulation Program): A free image drawing package, available for most operating systems (Mac, Windows, Linux)

-Within FSSH client, to use GIMP, invoke with

% gimp & (click OK through all the default settings)

-In the GIMP window, if you want to capture a windowed screenshot, go to:

*File 🡪 Acquire 🡪 Screenshot 🡪* and select *Single Window*

The cursor should become a crosshair, and then you may select the window you wish to capture. Once selected, you may save the captured image using:

*Right click mouse over image 🡪 File 🡪 Save as* and then simply select an extension (.bmp, .jpeg, etc.) and select a file name. Once saved, you may use FSSH secure transfer to save the new image file to your personal PAWS window.

[TOC](#TOC) [File Manipulation](#FileManipulation)

#### Changing file content with awk

Awk can be run as a series of instructions from within a script. I use if for simple operations that would be too troublesome to program in perl, C or any other language for that matter. A command-line-instruction using awk includes its principal instructions within single-inverted commas and braces

% awk '{ *instructions* }' infile > outfile

One example of great use for awk is to change the order of columns in a file. Create a file with three text columns separated by tabs. From the command line, enter:

% awk '{print $2, $3, $1}' infile > outfile

Commas will generate spaces between columns in the output file

Another very useful application for awk is to carry out some simple math on the columns of data:

% awk '{print $1 $2\*2 $3+3}' infile > outfile

Yet, another examples:

% awk ‘{print $3, $1}’ my\_name > my\_name\_reversed

“print $3, $1” is an instruction to print the contents of the 3rd column first, use a space, then print the contents of the 1st column. “my\_name” is the file that contains your name (for example Stevie Ray Vaughn) and the “my\_name\_reversed” would be printed as Vaughn Stevie.

A math operation example:

% awk ‘{print $3+1.1, $1/0.1}’ first\_file > second\_file

A simple command that tells the machine to take the contents of the 3rd column in “first\_file” and add 1.1 to it and then take the contents of the 1st column and divide by 0.1. These new values would be stored and created in “second\_file”.

A final use for awk is to do quick substitutions

awk '{gsub(/foo/,"bar");print}' infile > outfile

“gsub” is an instruction that is used to substitute one value or string of values for another. An example for the syntax would be:

% awk ‘{gsub (/3/, “6”); print}’ infile > outfile

Here, again, we’re asking the program to take a look at “infile”. The instructions inside the bracket tell the program to substitute any 3 (the value within the slashes) with a 6 (the value within the double quotations) and then print this into a new file called “outfile”.

Where can you get a tutorial?: [*http://www.vectorsite.net/tsawk.html*](http://www.vectorsite.net/tsawk.html)

Exercise

[TOC](#TOC) [File Manipulation](#FileManipulation)

### Vi (Visual Editor)

Vi has been insulted by many as an antiquated tool for working on a computer. But, regardless of the linux machine you have, and in the worst of field conditions ( from experience) vi will always work. There is a steep learning curve of a few weeks of practice, (like learning to type) but in the long run I save a lot of time. I save time because CLI are faster to enact and because vi has proven longevity so anything I learn today will still be useful 20 years from now. Many basic instructions in vi are inherited from the OS.

Some common instructions in vi include those shown in the following table:

|  |  |  |
| --- | --- | --- |
| Mode | Instruction | Purpose |
| Insert | i | To insert text and make general changes to text |
| Browse | :q | Close the file without making changes |
| Browse | :w | Save changes |
| Browse | :wq | Save changes, then close the file |
|  | ESC | Leaves Insert mode and takes you to Browse Mode |
| Browse | u | undo |
| Browse | dd | Delete current line and move into first buffer |
| Browse | x | Delete current character |
| Browse | 2dd | Delete 2 lines and mover them into the first buffer |
| Browse | dw | Delete next word |
| Browse | d2w | Delete next 2 words |
| Browse | yy | Paste the current contents of the buffer below |
| Browse | o | Open a line below the current position for editing mode |
| Browse | a | Append to the end of the line and enter the editing mode |
| Browse | p | Paste buffer |
| Browse | r | Replace current character by following character (enter a character) |
| Browse | R | Replace all following characters by the proceeding typed text  In order to leave this mode, enter R again |
|  |  |  |

|  |  |
| --- | --- |
| yy | copy |
| p | paste |
| u | undo |
| x | Delete character |
| escape | save |

#### Editing in vi

* Type in vi .login . You will now be looking at a screen with the .login file on it. Type in /set to search for variables beginning with ‘set’, because we’re looking for setenv, environmental settings.
* It should read setenv DISPLAY odyssey:0 So all displays will show up on the odyssey server on the main screen (0). You’ll have to change these variables so that it will display the images you pull down on your own personal machine (whether laptop or PC).
* To enter editing mode in vi, type in i. You can now being screwing things up. You’ll know you’re in i mode because insert will appear at the foot of the SSH screen. You can still navigate with the up and down arrows.
* At the beginning of the setenv line, add in a hash mark, #. This makes the line following it a comment. Comments are used in programs to provide information to programmers looking at the code.
* To leave editing mode in vi, hit Escape. To save your changes, type in :wq which writes the file (w) and then quits (q). You’re now ready to run Xming.

#### LINKS to sites with information on vi :

[Colorado State University Computer Science Department](http://www.cs.colostate.edu/helpdocs/vi.html)

[University of Hawaii at Manoa](vi.pdf)

## Perl {Hoffman, 2001 #3408} “**P**ractical **E**xtraction and **R**eport **L**anguage”

### Why use Perl?

There are certainly “better” ways to write code, but here are my reasons to use perl:

(1) It costs nothing, is mature and widely available

(2) Testing is quick; “on-the-fly”. Perl is an interpreted language which means that code is translated into the machine language while it is running one line at a time so that places where there are errors are easy to locate.

(3) Perl can easily incorporate shell programming scripts. Perl can be used as a “glue” to organize a computational workplace. Perl can be used to communicate between different modular command-line Open Source programs.

(4) Perl can be used for more complicated programs that require setting up functions or “sub-routines” that help keep complicated programs modular and simple

(5) Handling text files and their content is carried out more easily than with other programs

### When not to use Perl?

When there are many mathematical operations

When you know of an easier way that will save you time and frustration

When you are thinking about including a graphical interface to make it friendly to use

### Tutorials

A great place to start is to use the online tutorials in linux. Check this out:

% info perl

………….

perl Perl overview (this section)

perlintro Perl introduction for beginners

perltoc Perl documentation table of contents

Tutorials

perlreftut Perl references short introduction

perldsc Perl data structures intro

perllol Perl data structures: arrays of arrays

perlrequick Perl regular expressions quick start

perlretut Perl regular expressions tutorial

perlboot Perl OO tutorial for beginners

perltoot Perl OO tutorial, part 1 ………

perltooc Perl OO tutorial, part 2

perlbot Perl OO tricks and examples

perlstyle Perl style guide

Notes: Use the up and down arrow keys to move

Control C will get you out of any program

### Basic components of Perl

#### Input and Output

##### Printing Hello World

#!/usr/bin/perl

#This is my first program in perl

print (“Hello World\n\n”);

In the above example there are five things to note.

(1) The first line denotes the location of the perl binary

(2) From now on all items that are output to the screen will be included in parentheses and double inverted commas. Double-inverted commas permit Perl to interpret the different items. For example some items are read as text and others as “special characters” when needed. (Try out single commas just to see what would happen). If you want to null the value of a special character put a “\” before it. For example “[\\n](file:///\\n)” makes “\n” come out just like the characters you see. (Try it out).

(3) the “\n” is a shorthand code that means include a new line when the rest of the text is written out. There is a new line before the start of writing and there are two new lines after the start of writing.

(5) All lines except the first and the line commented out end with a “;” denoting the end of an instruction.

(4) The symbol “#”on the second line means that these words are informational for the reader and will not be considered by Perl to be a meaningful instruction.

##### [Reading](#perl_print_Hello) from and Writing to a file

If you want to read and write data to hard drive you must first tell the system you are ready to access a part of the hard drive. This is done by opening a “FILEHANDLE” or a file address. You must also provide a name. The FILEHANDLE should be closed when you are done reading or writing to the file.

Here is an example of opening a file:

#!/usr/bin/perl

open (FILE, “filename”) || die (“can’t open this file $!”);

$i=0;

while ($read = <FILE>) {

@line[$i] = $read;

$i=$i + 1;

}

$imax = $i;

close (FILE);

for ($i=0;$i<$imax;$i=$i+1) {

print (“@line[$i]”);

}

“$!” is a special operator indicating a system error has occurred.

“<>” is the line-reading operator which continues by itself until the end of the file is encountered

Line reading continues as long as the value of the “while” statement is true, i.e. as long as the content of the parentheses remains TRUE (=1).

Reading is quite straight forward except for the following:

(1) remember that lines of data may have invisible characters that you may want to remove

(2)

Here is an example of writing to a file:

#!/usr/bin/perl

$imax=3;

for ($i=0;$i<$imax;$i=$i+1) {

@line[$i] = $i;

print (“@line[$i]”);

}

open (FILE, “> filename”) || die (can’t open this file $!”);

for ($i=1; $i<3; $i=$i+1) {

print OUT @line[$i]

}

close (FILE);

Note that the only important difference between reading and writing is that we have a redirect sign “>” before the filename.

[Exercise](#Exercise_perl_file_write)

#### Data Types

Just as we saw in dealing with shell variables we distinguish between the value stored on a hard drive and the name associated with that number.

A perl **variable** is a **place** to store the value, which is called the **literal**.

For example:

#!/usr/bin/perl

#This is my second program in perl

$number = 2;

$output\_text = (“Hello world”);

Print (“\n$output\_text \n\n $number”);

When writing out text, note that text consists of individual characters strung together in a line, including minus signs, plus signs, spaces, tabs, end-of-line-characters, etc. A string of characters is just that, **a string**. In the example above we assign (“Hello world”) to the variable $output\_text.

[Perl-Data Types](#perl_IO)

Lists of Variables or Arrays

If you want to include various lines of texts it might be cleaner to break up the text into different segments. In order to handle this we can create a “list” of lines of text.

#!/usr/bin/perl

#This is my third program in perl

@output\_text[0] = (“Hello world\n”);

@output\_text[1] = (“I want to live\n”);

@output\_text[2] = (“I want to flourish\n”);

Print (“\n@output\_text[1] \n”);

**List variables** carry the “@” sign at the beginning of their name. The list is ordered starting at 0 and not at 1.

Yes, you could also write the list with a different syntax:

#!/usr/bin/perl

#This is my third program in perl

@output\_text = (‘Hello world\n’,’I want to live’,’I want to flourish’);

[Exercise](#Exercise_linux_password)

A list of variables is also known as an **array:**

#!/usr/bin/perl -w

#PURPOSE: describe perl arrays

@output\_text = (“ Four score\n”,”and seven years ago\n”,”our fathers landed\n”);

print(“@output\_text[2]\n”);

print(“$output\_text[2]\n”);

print(“The number of values in the array is: [@output\_text[$#output\_text] +1]\n”)

Is there a difference between the two outputs?

There are a couple special arrays which will need later when we write functions and perl programs that can interact with the user, that is they require input from the user such as a number or a file name on the command line : e.g.

%program\_name.pl input\_file\_name add\_this\_number output\_file\_name

The first variable is called **@ARGV** and keeps track of the order of the values that follow the name of the program above.

Another special variable **@\_** is needed to pass arrays to a subroutine (a sub-program)

Perl-List of Variabless or Arrays

Scalars

Scalars are single-value data types. That is, only one value is assigned to that variable and the value can be a string or a number. Scalars are indicated by a “$” sign at the beginning of the variable.

There is one special variable in perl that is useful to know. Commonly you will want to know the number of values your array. The length of your array or the number of values in your array would be equal to the largest index plus 1. For this purpose there is a special scalar variable in perl you can use. This special variable has a literal value equal to the last index in the array:

#!/usr/bin/perl -w

#PURPOSE: estimate array length

@output\_text = (" Four score","and"," seven years ago","our fathers landed");

$array\_size = $#output\_text + 1;

print("The number of values in the array is $array\_size\n");

print("The last of value stored in the array is:\n”);

print(“\t\t@output\_text[$#output\_text]\n");

Note have inadvertently we have introduced, briefly, how carry out some simple arithmetic from within perl.

[Perl-Scalars](#perl_scalras)

Hashes

Hashes represent a set of key/value pairs, but we will leave this for later.

[Perl-Hashes](#perl_hashes)

For-loop/Do-loop in perl

Do-loops (herein “for-loop”) are a term inherited from Fortran (and sh). In Perl there is a simple syntax to handle repetitive tasks that is very similar to C and Fortran, and Matlab. After all, computers ARE supposed to be used for doing repetitive tasks very fast. Here is how we do a loop:

#!usr/bin/perl

# NAME:

# PURPOSE: To show off for loops

$max = 10;

for ($i=0; $i<=$max; $i++) {

@output\_number\_array[$i] = $i+1;

}

for ($i=0; $i<=$max; $i++) {

print ("For index = $i \t value = \t @output\_number\_array[$i]\n ");

}

Inside the parentheses after the “for”, there are three instructions. The first instruction “$i=0” provides the START of the loop. That is, the first instruction is the first thing that is carried out in the loop. Remember this!

The second time the loop is run, the third instruction is carried out, i.e. the $i value is updated by adding 1 to the previous value. At that point the second instruction must be met for the calculations to enter the loop again. If the second instruction is not me then the loop is exited and the “$i” retains its previous value from the end of the last loop. To be safe, you can examine the value of $i when the loop is exited.

Note that we can work the index in reverse as well and that the values of “$i” can increment by more than just “1” each time.

[Exercise](#Exercise_perl_for_loop)

[Perl-for](#perl_IO) loop

#### Perl operators

Various symbols exist in perl that are very similar to operators in other programming languages. Operators can be of several types depending on whether you are dealing with NUMBERS or CHARACTER STRINGS.

**Arithmetic**

+ addition

- subtraction

\* multiplication

/ division

**Numeric comparison**

== equality

!= inequality

< less than

> greater than

<= less than or equal

>= greater than or equal

**String comparison**

eq equality

ne inequality

lt less than

gt greater than

le less than or equal

ge greater than or equal

**Boolean logic**

&& (and)

and (also and)

or (or)

! (not)

Not (also not)

**Miscellaneous**

= assignment

. string concatenation

x string multiplication

.. range operator (creates a list of numbers)

Many operators can be combined with a "=" as follows:

$a += 1; # same as $a = $a + 1

$a -= 1; # same as $a = $a - 1

$a .= "\n"; # same as $a = $a . "\n";

[Perl- operators](#perl_operators)

#### Conditional if

An if statement allows perl to pass judgement on two variables. If the judgement has a TRUE (1) outcome then the instructions inside the curly braces are carried out, otherwise (FALSE ; =0) the perl language jumps to the first line after the “If” statement.

An “if statement” in its shortest version looks as follows:

#!/usr/bin/perl

@value[1] = 1.1;

@value[2] = 1.0;

if (@value[1] >= @value[2]) {

print (“\You have entered the first set of instructions\n”);

}

else {

print (“\nYou have entered the second set of instructions\n”);

}

Exercise

[Perl-conditional](#perl_IO) if

System commands

All that you have learnt prior to perl regarding the linux OS and shell can still be used within perl. Say, for example you wish to generate the following working set of directories:

/home/loginID

/data /progs /images

/jpg /tiff

#!/usr/bin/perl

$HOME = (“/login/loginID”);

$DATA = $HOME

print(“\nMaking directories @directory[1] \n”);

system (“ \\

mkdir –p @directory[1] \\

“);

[Exercise](#Exercise_perl_SystemCommands)

[Perl-IO](#perl_IO)

## Exercises:

**Please e-mail me your answers in the form of an MS Word document**

### Directory structure and file locations

In this exercise use the ls instruction to outline the first four layers of directories under “/”. Name and describe three subdirectories within in each directory. In the example below the first three subdirectories of the second level are shown. Three suchdirectories are required for a 3rd and also a 4th level. What is contained in each directory. For example you will start with the following: **(50%)**

/ (1st level)

/var (2nd level)

/bin

/usr

While you are looking inside the directories, write down where the passwords seem to be kept **(25%)**. Also, where is the printer information kept, and where are files kept that are sent to the printer? **(25%)**

[Directory Structure of the Linux operating system](#DirectoryStructureInLinux)

### Do-loop

Write a shell that recursively adds 1 to an initial value. For example, start with 1 and then add 1 to that to produce 2. Add 1 to the value 2 to produce a value 3. Write out the answer each time. You need only write out 2 values at most. **(50%)** Call your file do\_loop.sh, leave it in your home directory, make it executable, editable and writable ONLY to you. **(50%)**

[*Repetitive tasks*](#repetitivetasks)

### Awk

Write a program that reverses a list of names in a file. First create a file that contains you first and last name separated by a tab. Next invert the order so that your last name appears first and your first name appears last. Make a copy of this program and e-mail it to me.

[Changing file content with awk](#awk)

### Perl lists

### Write out Lincoln’s Gettysburg address using lists.

### Perl for-loop

Write a perl “for-loop” that will generate an [1row x 10 columns] array:

24 21 18 15 12 9 6 3 0 -3

### Perl write to a file

Use the previous perl “for-loop” and write the array to a text file

On the first line write your name

Output three numbers per line

### Perl read from a file

Use the previous perl “write-to-a-file”.

Modify the previous perl program and use it to read the text file.

Output all the lines you read to the computer screen with a

neat format output.

### Perl **if** logical operator

Use the example in the instructors folder for this exercise. The example is called “perl\_if.pl” Modify this file so that the program also automatically outputs the name of the oldest child to the screen

[Conditional if](#perl_if)

### Perl system commands

Create a perl program(s) that generate(s) the following directory structure:

/home/”yourlogin”

/”yourProjectName”

/seismics /geomaps /wells /gravity

In each of the four previous directories create the following five (5) directories:

/data /images /pl /sh /matlab

In each of the immediately previous directories also create the following two (2) directories:

/081207 /010508

Finally in each of the immediately previous directories create:

/1 /2

Please call this program “yourname\_mkdir”

You should created about 160 directories as a result of this exercise.

(2) Repeat all the above exercises but simplify the program by including a “for loop “ to carry out the creation of files. Please call this program “yourname\_mkdir\_loop”

(3) Finally, create a program that will, in addition to step (2), change rwx permissions (chmod 655 “directory” ) of ALL the directories created above. Name this program “yourname\_mkdir\_chmod”

[Perl-System commands](#perl_SystemCommands)

### MATLAB - Basic Matrix Operations

Some of the following examples are taken from Trauth’s 2007 book, the Mathworks online tutorial and a great online tutorial at the following website

http://www.stanford.edu/~wfsharpe/mia/mat/mia\_mat3.htm

This is a demonstration of some aspects of the MATLAB language.

First, let's create a simple vector with 9 elements called a.

a = [1 2 3 4 6 4 3 4 5]

a =

1 2 3 4 6 4 3 4 5

Now let's add 2 to each element of our vector, a, and store the result in a new vector.

Notice how MATLAB requires no special handling of vector or matrix math.

b = a + 2

b =

3 4 5 6 8 6 5 6 7

Creating graphs in MATLAB is as easy as one command.

Try the following three different cases to see how the color and marker in the graph can be changed.

plot(b)

plot(b,’\*’)

plot(b,’r\*’)

### Syntax in MATLAB

Matrix stands for matrix laboratory, which is what MATLAB does best.

Let’s now create a multi-dimensional array:

A = [ 1; 2; 3]

B =[4; 5; 6]

Try adding, subtracting and multiplying these two arrays. Examine the error when you try to multiply them.

At this point you will need to transpose one of the arrays if you want to carry out a matrix multiplication

A’

Exercise: Multiply element-by-element by each other.

Exercise: So how do we plot a function such as:



### MATLAB File input and output

Let’s create some data and read it back:

% Sample ID Percent SiO2 percent Ca

1 53 2

1 50 3

1 59 1

**Example commands in MATLAB**

Type commands on the command line.

To view command history, type “h” or select command history under the desktop tab.

You can open several windows at a time to view different figures.

To tell MATLAB to search for files in directories outside of MATLAB, need to set up pathway manually:

File 🡪 Set Path 🡪 Add Folder

To run a program:

Debug 🡪 F5 Run

To create a vector array:

A = [1 2 3] or B = [1; 2; 3]

w/o semicolon will create a 1 row x 3 column matrix:

A = 1 2 3

w/ semicolon will create a 3 row x 1 column matrix:

1

B = 2

3

To transpose a vector, type the variable name followed by ‘ (as in A’ or B’)

Transposing a 1r x 3c vector will create a 3r x 1c vector and vice versa, which can be useful to make matrices compatible for certain algebraic operations

Type the variable name to spit out the array stored in memory

Type clear followed by the variable name to clear memory (as in “clear A”)

Or type “clear all” to clear all variables stored

Type “size A” to view the *dimensions* (# of rows and columns) in array A

Type “length A” to view the *length* (total # of values) of array A

To view all stored variables:

Desktop 🡪 Workspace

**Plotting Vectors and Simple Matrix Algebra**

To plot one vector vs. another type “plot (A, B)”; the vectors must have the same *length* but not necessarily the same *dimensions*

To experiment with line style and color in plots try typing (the possibilities are endless):

Plot (A, B, \*)

Plot (A, B, --)

Plot (A, B, r)

Plot (A, B, :r)

To add two matrices simply type “A + B”; the vectors must have the same *dimensions*

i.e. you can add a 1r x 3c array and a 1r x 3c array

To multiply two matrices simply type “A \* B”; the vectors must have the same *transposed dimensions* as each other

i.e. you can multiply a 1r x 3c array and a 3r x 1c array

To multiply two vectors w/ the same dimensions, typing “A .\*B” will override the rules of matrix algebra

**Creating Files in MATLAB**

To create new file, click New M-File (blank page icon at top left corner)

% sign tells editor to ignore that line (just as # sign in vi editor)

To save file, click save as and select desired destination

To load file, type “load filename” on command line

Type just the name of the file on the command line to view data stored in the file

To plot data in a file, first assign variable names to a particular row or column, i.e. for a file named “geochem”:

Ca = geochem (:, 3) 🡪 all rows in 3rd column

Si = geochem ( :, 2) 🡪 all rows in 2nd column

Plot (Si, Ca)

Useful MATALB FUNCTIONS

|  |  |
| --- | --- |
| NAME | Purpose |
| size (variable) | gives you the amount of rows and columns in an array |
| length(variable) | gives you the length of an array |
| x = [value1,value2] | assigns values to an array |
| x \*x | multiply matrix arrays |
| x .\* x | multiply element-by-element in two arrays for all elements |
|  |  |
|  |  |

### MATLAB – creating a spike (Stealth Fourier Theory)

#### Aim

To review the following MATLAB functions: (1) size, (2) length, (3)plot, (4)array creation, (5) axis values, (6) for loops

Exercise: Create a spike by adding up a very large number of cosine curves

#### Theoretical Background

Theoretically a spike is also called a delta function:





The amplitude spectrum of a spike is flat.

#### Seismological Applications

Seismologists can create a seismic source by (1) generating an **impulse** of energy (all at once) or by (2) gradually introducing the energy into the ground across a range of frequencies.

#### Procedure

In order to learn more about a Delta function let’s begin by examining the following function:



In MATLAB, let’s begin by examining the effect of changing f(requency), **t**(ime) and **ph**(ase) on different plots.

STEPS

* Plot the function for f=1 Hz
* Overplot the function for f=2 Hz
* Do the same for 4 Hz, up to 10 Hz
* Apply a positive phase and a negative phase of pi/2 radians

To do the following lesson, you should download spike.m from ctg08/progs/matlab on the odyssey server.

Once downloaded, you need to remember to set the path via the file menu so that MATLab knows how to find your file.

The goal of the lesson is to provide an example of Fourier Theory – that many frequencies of a signal to summed together act as a single pulse. For seismic, it’s like the equivalence of using an explosion or using a truck to put in varying frequencies into the ground which sum to basically be an explosion (or chirp).

**Important Functions** (code is in bold)

**%** is used before a piece of code or sentence to “comment” it out

How to make arrays **f = 1:10** will create an array of 1 through 10 by 1s, 1 row 10 columns

**F = 1:.5:10** will create an array of 1 through 10 by .5s

**Plot(x,y)** plots one array with respect to another

**Hold on;**: Keeps graphs on the screen; allows more than one graph to be displayed

**Hold off;**: Stops graphs from overlapping

For loops: **for f=1:2** will run a for loop incrementing f by 1; for f =1:.1:10 will run a loop incrementing f by .1. To end a for loop, type end.

For example:

for f=1:2

%code inside of loop

end

**size (variable)** provides the length of a multidimensional array

**length (variable)** provides the length of a 1D array

It is also important to note, to multiply by a constant, put a period after the variable letter **f.\***

**clear all;**  clears all stored variables (and arrays)

As a reminder, adding a semicolon after a piece of code will hide it from the command window, but leaving it will show it (ie if you want to see what’s going on don’t type a semicolon).

**The program using cosines**

% create your time array

t = -10:.05:10

% grab the size of the time array and initialize your y array with 0s

b = size(t)

y = zeros(b)

% run a for loop to add multiple frequencies of cosine together

for f=1:200

y1 = cos(2 \* pi \*f .\* t)

y = y + y1

end

% plotting

plot(t,y/200)

axis([-1 1 -1 1])

press **F5** to run your program.

Exercise: What do you get when you add up a very large number of cosine curves?



## References

Albing, C., Vossen, J., and Newham, C., 2007, bash cookbook: Sebastapol, CA, O"Reilly, 598 p.