CLEO-101 Lecture 2: Getting Setup And Ready to Go!

Richard Gray
Emilie Phillips
June 8, 2005
Goals For Today:

*Setup Your Account

*Understand Software Releases

*Create, Compile, Run Code

(break or end)

*Extras if we get time
Account Setup: What’s the Password?

*You could potentially have up to 4 passwords here. To read about them, see http://www.lns.cornell.edu/public/COMP/account_pass.html

*VMS password: Use command `SET PASSWORD` on lns61 or lns62 VMS machines.

*Unix password: You will use unix every day. On unix terminal use command `kpasswd` to set password

*Network Principal: Use this to access secured web pages and email. Change on unix using `knetpw`

*Windows: Some People will use Windows to access Unix. Others will only use windows to use the scanner, etc. Change by pressing `CTRL-ALT-DELETE` after logging in, and click on “change password.”
Account Setup: Unix/Linux
You will be using the Unix or Linux operating system.

*Learn basic commands: ls, cd, mkdir, rm, ln, more, cat etc..

*Unix/Linux is case sensitive

*Edit files with: emacs, vi, pico etc... For emacs see:

http://www.lns.cornell.edu/~asner/private/cleo101/day02/intro_emacs_short.html
http://www.lns.cornell.edu/~asner/private/cleo101/day02/intro_emacs_long.html

*Can read email with pine.

*For More unix commands see:

http://www.lns.cornell.edu/~asner/private/cleo101/day02/unix.html
Account Setup: UNIX (hurray!!)

Here are some basic UNIX commands:

*ls lists contents of directory
example

prompt> ls
myfile.txt anotherfile.txt whats_in_here.txt MyDirectory dontlook.txt
CToMeIAmADirectory PhysicsStuff ADocument.pdf

*cd change do a different directory
example

prompt> cd MyDirectory
prompt> ls
cléo.txt c.pdf is.pds for.rp me.txt

*mkdir create a new directory
example

prompt> mkdir MyNewestDir

*pwd “print working directory” tells you which
directory you’re in.
example

prompt> pwd
/home/gwbush
prompt> cd MyNewestDir
prompt> pwd
/home/gwbush/MyNewestDir

*cp copy one file to another.
example

prompt> cp original.txt copy.txt
Account Setup: UNIX (more!!)

*cp -r Allows you to copy directories

Example

prompt> cp -r ADir CopyDir

*rm Remove files, rm -r remove “recursive” directories, and everything in it.

*rmddir Remove directories, only when they’re empty.

*ps List the processes (programs) you’re running on the computer.

Example

prompt> ps
  PID TTY  S   TIME CMD
  30297 tty3  I + 0:10.48 pine
  14442 tty6  S   2:46.67 emacs cleo101_talk.tex
  28102 tty5  I + 0:00.73 krlogin lnx102

*kill Kills a process (program) you want to stop.

Example

prompt> ps
  PID TTY  S   TIME CMD
  14442 tty6  S   2:46.67 emacs cleo101_talk.tex
prompt> kill -9 14442
prompt> ps

*echo Writes argument to screen.

Example

prompt> echo "I love CLEO-c"
I love CLEO-c
**Account Setup: UNIX (Even more!!)**

*more, less, cat* They open up text files and display them to the screen. Each one works a little differently. Experiment.

**example**

```
prompt> more DontLook.txt
I told you not to look
Why are you looking
bla bla bla bla bla
prompt>
```

*ln* Creates a shortcut to a file or directory in another directory or on another disk.

**example**

```
prompt> cd /this/is/a/really/long/path/name
prompt> more hi.txt
hello world!
prompt> cd /home/gwbush
prompt> ln -s /this/is/a/really/long/path/name/hi.txt hi.txt
prompt> more hi.txt
hello world!
```

*man* VERY IMPORTANT!! Gives you documentation on most Unix commands.

**example**

```
prompt> man kill
kill documentation
bla bla bla bla bla
prompt>
```
Account Setup: UNIX (Episode 3: Revenge of the Shell)

*diff* Shows the differences between files.

```bash
prompt> cp afile.txt newfile.txt
prompt> diff afile.txt newfile.txt

prompt> diff afile.txt dfile.txt
< I am in afile.txt and not in dfile.txt
> I am in dfile.txt and not in afile.txt

```

*grep* Looks for specific strings in files.

```bash
prompt> grep "'I am'" afile.txt
I am in afile.txt and not in dfile.txt
```

*quota* Checks your disk space quota. You will get an email if you go over your quota.

```bash
prompt> quota
Disk quota for user rcg24
   Filesystem  blocks quota  limit  grace files quota  limit  grace
       /dev/vg/cleocg/lv1     168221 200000  220000     15489  20000  220000
```

*ps2pdf* convert postscript files to pdf files. *pdf2ps* convert pdf files to postscript files.

```bash
prompt> ps2pdf afile.ps afile.pdf
```

*chmod* change permissions on a file. Can make a script executable, or protect a file from being over-written.
Account Setup: UNIX (Exciting Combinations!)

You can combine commands with pipes “|”. You send the output of one command to the input of the other command.

examples

prompt> ls
file1 file2 file3 dir1 dir2 dir3
prompt> ls | grep file
file1
file2
file3
prompt> ls | grep dir
dir1
dir2
dir3

You can also redirect output to a file with “>” or “>>”.

examples

prompt> echo ‘‘welcome to cleo’’ > welcome.txt
prompt> more welcome.txt
welcome to cleo
prompt> echo ‘‘welcome to cleo’’ > welcome.txt
prompt> more welcome.txt
welcome to cleo
prompt> rm welcome.txt
prompt> echo ‘‘welcome to cleo’’ >> welcome.txt
prompt> more welcome.txt
welcome to cleo
prompt> echo ‘‘welcome to cleo’’ >> welcome.txt
prompt> more welcome.txt
welcome to cleo
welcome to cleo
Account Setup: UNIX (Jokers are Wild)

“Wild Cards” are very useful

** The “*” can be used as a wild card when looking for files

example

```
prompt> ls
file1.txt file2.txt file3.txt file1.dat file2.dat file3.dat
prompt> ls *.txt
file1.txt file2.txt file3.txt
prompt> ls *.dat
file1.dat file2.dat file3.dat

*You can give a range of “wild card” values too.

example

prompt> ls
a1.txt b2.txt c3.txt d4.txt
prompt> ls [a-c]*.txt
a1.txt b2.txt c3.txt
prompt> ls *[2-4].txt
b2.txt c3.txt d4.txt
```
**Account Setup: Other Unix Utilities**

Here are other Unix Utilities that are very useful. When you get time, google for them on the internet, or use **man** and learn how to use them.

* **awk** string manipulation in scripting

* **sed** string manipulation in scripting

* **tar** putting large numbers of files and directories into one file

* **cvs** software development management system.

* **emacs** or **vim** text editing.

* **find** What is lost will be found (finds files)

* **gmake and Makefiles** Helps you compile code. Learn more when you have time.
Account Setup: Disk Space

*You have a limited home directory of 200MB, backed up daily

*Put your source code, logs, anything you can’t afford to loose here

*Put any large files that you can easily recreate (i.e. compiled code) somewhere else!!
  *temporary space available on /cdat/tem directory

*Cornell has space on the ’daf’ disk farm. Contact Brian Heltsley if you need this.
Account Setup: EMAIL

*You have two email addresses! One for VMS system and one for UNIX. You don’t want to have to log into VMS every day. So, set mail forwarding so that VMS mail gets forwarded to your UNIX mail.

*Please see instructions on the following webpage:
http://www.lns.cornell.edu/public/COMP/mail_forward.html
Account Setup: Mailing Lists

*There are several mailing lists that you will want to be on.

*Manage the lists you are on using listar. Simply email the automated listar system commands at: listar@mail.lns.cornell.edu

*listar will execute the first valid command it finds in the subject or body of the email.

*Email listar with the command “help” in the subject line. You will be emailed a help file with a list of commands.
**Account Setup: Mailing Lists (2)**

Here are some lists that you should consider subscribing to.

*cleo-grads*: This list is for CLEO graduate students. Summer REU students should also add themselves for the summer. In particular, tells you when we have free food in Wilson.

*cornell-grads*: For Cornell LNS graduate students.

*c3-sw*: Get notices relating to CLEO software.

*Ask your advisor if there are other lists they want you to be on. Example: If your project were on Linear Collider Cosmology studies, you would want to be on “lc-cosmo” list.*

*See list of mailing lists on computer group website:*

http://www.lns.cornell.edu/restricted/COMP/mailing-lists.html
Account Setup: Problem?

If you happen to have a **hardware** (not software) problem, then contact the computer group service. You can send a service request on the computer group at their web page here:

http://www.lns.cornell.edu/public/COMP/RequestForm.html

For problems with cleo software:

*First ask your fellow students and CLEOns near you or the next office over. (Most of us don’t bite. I only kicked someone in the head once, and that was on accident)*

*Second, email “c3expert”, c3expert@mail.lns.cornell.edu*
Account Setup: The Shell

*The program that Unix runs to accept your commands and execute them is called a “shell”.

*There are different shells to choose from (tcs, bash)

*The basic commands are the same (ls, cd, mkdir etc.) but there are important differences.

*People at CLEO typically use either tcs or bash... The default shell for new cleons is bash.
Account Setup: Set Enviroment Variables

*The shell has a number of “enviroment variables” set up to tell different processes (programs) where things are, the computer you’re using, etc.

*Type the unix command `env` to get a list of all the enviroment variables already set in your shell.

*You can use the “echo” command to see what these are. (Note the $ in example)

example

```
prompt> echo PATH
PATH
prompt> echo $PATH
/home/gwbush/bin:/usr/bin:/usr/bin/X11:/usr/local/bin:/usr/local/bin/X11
prompt> echo HOME
HOME
prompt> echo $HOME
/home/gwbush
```

*You will want to set enviroment variables so that your shell can take advantage of CLEO software
Account Setup: Set Environment Variables (2)

*In **bash** do the following:

```bash
export CLEO3DEF_SILENT=1
./nfs/cleo3/offline/scripts/cleo3logins
./nfs/cleo3/offline/scripts/cleo3defs
c3rel 20050316_FULL
export PATH=$PATH:$ROOTSYS/bin
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
```

*In **tcsh** do the following:

```bash
setenv CLEO3DEF_SILENT 1
source /nfs/cleo3/offline/scripts/cleo3logins
source /nfs/cleo3/offline/scripts/cleo3defs
c3rel 20050316_FULL
setenv PATH $PATH:$ROOTSYS/bin
setenv LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
```

* **c3rel** is a special CLEO command that sets the version (release) of cleo software to use. More on this later.
Account Setup: Add to .login or .bashrc

*You don’t want to have to type the enviroment setup every time. It will get really annoying really fast if you do.

*Files in your home directory .bashrc for bash shell or .login for tcsh will automatically get executed each time you log in.

*Add the lines from the last slide to these files so you don’t have to type it every time.
Account Setup: Printing

*You will probably want to print things on occasion.

*To get a list of possible printers do:

```
prompt> lpstat -a
```

*Then, to print either text files or Postscript files, and you want to print to the Wilson room 320 printer, do the following:

```
prompt> lpr -P w320_xrx_n4525 MyFile.ps
prompt> lpr -P w320_xrx_n4525 anotherfile.txt
```

*Can set default printer by setting environment variable “PRINTER” in .bashrc or .login

*DO NOT TOUCH THE PRINTERS. NEVER NEVER NEVER. DO NOT LOAD PAPER. DO NOT PRESS ANY BUTTONS. NEVER, NEVER, NEVER. If a printer needs attention, please page the batch operator on the beam phones.
Understanding Software Releases

*In CLEO there are over 780 “offline” software packages

*This includes analysis code, code for delivering data and information, code for generating Monte Carlo Simulated Data

*These packages are always evolving as improvements are made and the goals of the software change (i.e. switching from CLEO3 to CLEO-c)

*The code that you will make and modify will depend on many of these packages.

*As things change, your code may break because of a change someone made to another package
Understanding Software Releases: Fixed Release

*Periodically we create “Fixed Releases”

*A fixed release contains all of the software as it was on a given day.

*Experts have done their best to make sure everything in a fixed release works.

*Unless significant bugs are found, they are stable.
Understanding Software Releases: Selecting a Release

*Use c3rel to choose which release you want to use

c3rel
c3rel current
Mar04_04_MC
20040610_FULL
20040622_FULL
20041008_MCGEN
20040803_FULL
20040423_FULL
20041008_FULL
20041104_P2
20041104_MCP2
20041201_FULL
20050120_FULL
20050214_MCGEN
20050316_FULL
20050417_FULL
20050525_MCGEN

c3rel 20050316_FULL

cleo3def - Setting Up Cleo3 Environment
   Using release 20050316_FULL
Understanding Software Releases:
Types of Releases

* Releases made within the last year or two will be type FULL, MCGEN, P2 or MCP2

* **FULL** releases have all the CLEO software. Use these in your day to day work.

* **MCGEN** releases have only the code needed to generate Monte Carlo Simulated Data at “cleog” stage

* **P2** releases have only the code needed to process real data with the level of analysis called “pass 2”.

* **MCP2** release intended to do analysis “pass2” stage on Monte Carlo Simulated Data. Date in Release signifies date of “pass2” code. MC code was most recent when release was made.
Understanding Software Releases: current and development

*current and development releases are NOT fixed!

*development is a testing area for the newest changes to the cleo software. It recompiles and changes each night, and is often broken.

*A new current release is created everytime the code in development passes all tests. It changes regularly. Sometimes it’s broken.

*only software experts, working on improving important code, should use current or development.
Understanding Software Releases: Someday

*You will someday be a software expert. Software experts use \texttt{cvs} to check in new code, and changes to code into a master repository.

*When your new code is ready, check it in with \texttt{cvs}. Then, use the script \texttt{c3tag} to “tag” the change as ready for \texttt{development}. \texttt{development} will automatically build the latest, tagged changes the following night.

*If there are problems with your code, you will be automatically emailed by the nightly build scripts.

*If you are not yet an expert, or you just want to check code out, please use \texttt{cleo3cvs}. \texttt{cleo3cvs} has all the proper paths automatically set, and won’t let you alter the master repository.
Understanding Software Releases:
Flavors of Unix

*CLEO has used 4 types of operating systems: **OSF, Solaris6, Solaris8, Linux.**

*Code compiled on one flavor of Unix will typically **NOT WORK** on another flavor.

*For a given fixed release, there are multiple sets of binary files for each flavor that was available when the code was created.

*OSF and Solaris6 can only be used on CLEO2 and CLEO3 analysis. They are obsolete for CLEO-c, and the machines running these platforms are showing their age. (The former race horses are on their way to the puppy chow factory).

*YOU will use **Linux**
Understanding Software Releases: Path Names

*Depending on what machine you are on, a given directory path may not take you to the same place.

*Offline Fixed Release binaries are kept at: 
/nfs/cleo3/Offline/rel/RelName

*If you are on solaris /nfs/cleo3 takes you to 
/nfs/solaris2/cleo3 where you will find solaris binaries.

*If you are on Linux /nfs/cleo3 takes you to 
/nfs/linux/cleo3 where you will find linux binaries.

*Also, on Linux /cdat/tem brings you to /cdat/linux/tem, and on solaris /cdat/tem brings you to /cdat/solaris2/tem.
Creating Code: Processors and Producers

*Most of the software you create will be either a processor or a producer that will be run inside of another program called suetz.*

*There will be more about suetz in tomorrow’s lecture. Right now, just think of suetz as a program that helps your code work with the rest of CLEO’s code.*

*producers add information to the data when we run over it.*

*processors will perform your analysis, filter out data you’re not interested in, create ntuples or histograms to look at in root or paw, etc.*

*New cleons will mainly deal with processors*
Creating Code: Source and Build Areas

*As was mentioned earlier, you will want to keep your code in your home directory (which is backed up every night), and your compiled binary files somewhere else.

*Also, you will want to have separate build areas for different Unix platforms (i.e. **Solaris** or **Linux**).

*In the future you may find yourself doing multiple projects, each in a different fixed release. So, you’ll also want different build areas for different releases.
Creating Code: Source and Build Areas (2)

*When you get to your computer, log in and do the following (change gwbush to your name)

```
prompt> cd /home/gwbush
prompt> mkdir my_src
prompt> cd /cdat/linux/tem
prompt> mkdir gwbush
prompt> mkdir gwbush/build
prompt> mkdir gwbush/build/20050316_FULL
prompt> mkdir gwbush/build/20050316_FULL/Linux
prompt> cd /home/gwbush
prompt> ln -s /cdat/linux/tem/gwbush/build build
```

*You created a place to store your code, and a place on the tem disk to build the code and store your binaries.

*You also created a “softlink” in your home directory using the unix ln command.

*Also, we added subdirectories for the release we are using, and the unix platform we are using.
Creating Code: Using ’mkproc’

*We will be using C++ as a programming language. Fortunately, if you use the CLEO script mkproc, you don’t have to be an expert right away in order to start looking at data.

*mkproc will create a processor template for you. It can even add in example code snippets for doing certain types of tasks. If you want to get a list of all the different types of example code snippets mkproc can give you, use the “-list” option. See below.

```
prompt> mkproc -list
known example types:
   dchain
   doubletag
   dedx
   dtag
   eventshape
   histogram
   kshort
   mc
   pi0
   rich
   shower
   track
   tuple
   tuplevector
```
Creating Code: Your First Processor

* `mkproc` will let you combine the options in the list so you can have multiple example code snippets in one processor.

* Note that `mkproc` is release dependant. The code that `mkproc` creates for an old release may not work in a new release. The reverse is also true.

* Now, create your very first processor!! (I can tell you are excited). Lets make one that will look at both tracks and showers, and makes a histogram (Will use histograms in lectures to come). See example (change gwbush to your name)

```
prompt> c3rel 20050316_FULL
prompt> cd /home/gwbush
prompt> cd my_src
prompt> mkproc -track -shower -histogram MyFirstProc
```
Creating Code: MyFirstProc

*Lets examine what `mkproc` just did. It created a directory MyFirstProc. Now list the contents of that directory.

```
prompt> ls MyFirstProc
Class  Doc  Instantiate  Makefile  MyFirstProc  README.MyFirstProc  Test
```

*The most important objects to consider right now are the Class directory, the MyFirstProc subdirectory, and the Makefile file.
Creating Code: Class directory

*Do an ls on the Class directory:

```
prompt> ls MyFirstProc/Class
MyFirstProc.cc MyFirstProc_DONT_TOUCH2.cc MyFirstProc_DONT_TOUCH.cc
```

*The first task is to determine which of the above 3 files are okay to touch. If you guessed it was that file that didn’t have “DONT TOUCH” in its name, then you are correct. Give yourself a pat on the back.

*The file **MyFirstProc.cc** is the class definition of your Processor, and is the meat of your analysis. Use emacs to look at the file, but don’t change anything yet.

```
prompt> emacs MyFirstProc/Class/MyFirstProc.cc &
```

*Note, the & at the end of command tells the shell to run emacs in the background. Otherwise, you’ll loose the ability to interact with the command prompt when the emacs window pops up.
Creating Code: MyFirstProc.cc

*Looking at MyFirstProc.cc you’ll notice different sections of code begining with “MyFirstProc::”. These show different parts of the class that will get executed at different points when suez runs. If you are already familiar with C++, these are the methods of the MyFirstProc class

*For example, “MyFirstProc::hist_book” heads the section of code that tells suez what to do if you want to set up a histogram or ntuple file to write data out to.

*The section “MyFirstProc::event” will get executed everytime suez processes a data event. This is where you will put much of your analysis code.
Creating Code: MyFirstProc directory

*Now look at the subdirectory MyFirstProc. Do an “ls” and find out what is in there.

prompt> ls MyFirstProc/MyFirstProc
MyFirstProc.h  Template

*This directory contains the include or “.h” files for the MyFirstProc Class. You will need to modify this file to add data members, such as pointers to histograms or ntuples that you want your processor to fill.
Creating Code: the Makefile file

*The Makefile tells the program “gmake” how to compile the processor so that suez can load it and run it.

*A beginning Cleon will want to pay particular attention to the part in the Makefile that defines “CLEO3_LIBS”

```
CLEO3_LIBS :=  ProxyBind \
                Processor \
                CommandPattern \
                Navigation \
                TrackShowerMatching \
                C3cc \
                TrackDelivery \
                TrackRoot \
                KinematicTrajectory \
                ProxyBind \
                CleoDB \
                StorageManagement \
                DataHandler \
                ToolBox \
                Utility
```

*CLEO3_LIBS gives a list of other cleo packages that your code needs to build properly. For example, if you wanted to read in Monte Carlo information, you would have to add “MCInfo” to the “CLEO3_LIBS” list.
Creating Code: Now Build it

* Now we will compile/build the class. First, since building code can create a lot of network traffic, log into the linux machine that is dedicated to compiling code, lnx134. Set special enviroment variables to point to where our source is, and where we want to build it.

*For a list of available machines see:

www.lns.cornell.edu/restricted/CLEO/CLEO3/soft/hosts.html

*For bash:

prompt> kinit
prompt> klogin lnx134
prompt> export USER_SRC=/home/gwbush/my_src
prompt> export USER_BUILD=/home/gwbush/build/20050316_FULL/Linux
prompt> export USER_SHLIB=$USER_BUILD/shlib
prompt> c3rel 20050316_FULL

*For tcsh:

prompt> kinit
prompt> klogin lnx134
prompt> setenv USER_SRC /home/gwbush/my_src
prompt> setenv USER_BUILD /home/gwbush/build/20050316_FULL/Linux
prompt> setenv USER_SHLIB $USER_BUILD/shlib
prompt> c3rel 20050316_FULL

*For an easier way to modify enviroment variables, use (don’t look) Emilie Phillips’s script cleo_init.sh on cleo101 day02 webpage.
Creating Code: Now build it (2)

*Finish directory structure in build area:

```
prompt> cd /home/gwbush/build/20050316_FULL/Linux
prompt> mkdir MyFirstProc
prompt> cd MyFirstProc
prompt> ln -s /home/gwbush/my_src/MyFirstProc/Makefile
```

*To build **MyFirstProc**, we just need to create a **MyFirstProc** directory in the build area, then add in it a softlink to the Makefile in the source area.

*Now, type “gmake” in the **MyFirstProc** build directory.

```
prompt> cd /home/gwbush/build/20050316_FULL/Linux
prompt> cd MyFirstProc
prompt> gmake
```

* A large amount of text should scroll by as the code compiles. As long as you don’t see “Error”, you’re probably okay.

*Notice that gmake also create directories **lib** and **shlib** in the build area. They hold different types of binaries. The files in **shlib** are the binaries that **suez** will load and run.
Creating Code: It Compiled!

* Congratulations! You’ve created and compiled your first processor. Before we try to run it in suez, I thought I’d mention a few things about building code.

* `gmake clean`: If you change the list of libraries in “CLEO_LIBS” in the Makefile, you might have to do the command “gmake clean” in the build directory before things will compile correctly. The “gmake clean” effectively removes the dependency files `.d`, and the binaries that the build created. You’ll start from scratch.

* `.release` gmake will warn you if you try to build with a different release than what you’ve already used in a given build area. It will create `.release` files in the build area that give the name of the first release you used in that area.
Creating Code: Building Another Way

The way I just showed you to build works just fine if you only have a few packages in your build area. However, imagine if you had 100 of them for your project. Separately creating, and going to each sub-directory and typing “gmake” is going to get really annoying really fast.

*You want a main Makefile for the whole build area. Do the following:

```
prompt> cd $USER_BUILD
prompt> cp $C3_LIB/Gmake/userMakefile Makefile
```

Now, if you type “gmake” in the $USER_BUILD area:
*It will create a subdirectory called Linux (the OS).
*all packages in source area will automatically have subdirectories made for them here,
*They will all be compiled in the right order.
Creating Code: Building Another Way (2)

*If you use this main Makefile, you should change your $USER_SHLIB from "$USER_BUILD/shlib" to "$USER_BUILD/Linux/shlib", then do the “c3rel” command again.

*If you only want to compile one of your packages, you may do:

```
prompt> cd $USER_BUILD
prompt> gmake MyFirstProc
$```
Creating Code: Now Test It

*In order to test your processor, start `suez` and input the following commands.

```
prompt> suez
Suez> default prompt off
Suez> module sel EventStoreModule
Suez> eventstore in 20050316 physics all
Suez> setup_analysis
Suez> proc sel MyFirstProc
Suez> prod lss
Suez> proc lss
Suez> go 1000
Suez> exit
prompt>
```

*To save yourself typing in the future, create a tcl script with all of the above commands in it. In emacs (or your favorite text editing program) create the file `MyFirstProc.tcl` and add into it the above commands.

see: http://www.lns.cornell.edu/~asner/private/cleo101/day02/MyFirstProc.tcl

*Now to run this script in suez you can simply type:

```
prompt> suez -f MyFirstProc.tcl
```

or

```
prompt> suez
Suez> run_file MyFirstProc.tcl
```
Creating Code: Did it Work?

Check the output to the screen from running your processor:

```plaintext
%% INFO-Processor.MyFirstProc: track 1
%% INFO-Processor.MyFirstProc: momentum: (0.863102,4.36026,-1.51443)
%% INFO-Processor.MyFirstProc: track 2
%% INFO-Processor.MyFirstProc: track 3
%% INFO-Processor.MyFirstProc: momentum: (0.278359,1.57132,1.63253)
%% INFO-Processor.MyFirstProc: track 4
%% INFO-Processor.MyFirstProc: momentum: (-0.67448,-2.07016,-0.703466)
%% INFO-Processor.MyFirstProc: track 5
%% INFO-Processor.MyFirstProc: shower 1
%% INFO-Processor.MyFirstProc: energy: 2.04272
%% INFO-Processor.MyFirstProc: shower 2
%% INFO-Processor.MyFirstProc: energy: 0.307408
%% INFO-Processor.MyFirstProc: shower 3
%% INFO-Processor.MyFirstProc: energy: 0.300645
%% INFO-Processor.MyFirstProc: shower 4
%% INFO-Processor.MyFirstProc: energy: 0.218363
%% INFO-Processor.MyFirstProc: shower 5
%% INFO-Processor.MyFirstProc: energy: 0.0350236
%% INFO-Processor.MyFirstProc: shower 6
%% INFO-Processor.MyFirstProc: energy: 0.0322341
%% INFO-Processor.MyFirstProc: shower 7
%% INFO-Processor.MyFirstProc: energy: 0.0156086
```

*Above is an example of 1 event. We printed out the momentum of tracks and the energy of the showers. The units are GeV.*
Goals For Today:

Hopefully, You Learned:
*Setup Your Account

*Understand Software Releases

*Create, Compile, Run Code

Now Go to your computers, and try to go through the different exercises outlined in this talk. GOOD LUCK!
**EXTRA**

The Rest of this talk is extra in case we get time and students want to keep going.

*A little Looking at C++*

*How to find out more about the C++ objects*

*Recognizing “Undefined Symbols” errors.*

*A simple example of checking out and modifying code.*
Creating Code: MyFirstProc.cc
Again

*When you get to your computers, run emacs on the file MyFirstProc.cc and look at the code.

*Find the parts of the code that make the print statements. (They will be in the event method of the class).

example:

```cpp
report( INFO, kFacilityString )
    << "  momentum: "
    << (*pionFit).momentum()
    << endl;
```

*Note that we use report, rather than the standard C++ cout

*Now add some print statements of your own. Re-compile, and run the code to see them.
Creating Code: MyFirstProc.cc, Iterators

*Notice the “extract” calls.
example:

```cpp
FATable< NavTrack > tracksTable;
extract( iframe.record( Stream::kEvent ), trackTable );
```

*This extracts a table of tracks `trackTable` from the data flow (frame). It is a list of pointers to objects in memory that has all the information for a given track. We loop through these using iterators.
example:

```cpp
FATable< NavTrack >::const_iterator trackTableBegin = trackTable.begin();
FATable< NavTrack >::const_iterator trackTableEnd = trackTable.end();
// Loop over tracks.
for ( FATable< NavTrack >::const_iterator trackItr = trackTableBegin; trackItr != trackTableEnd; ++trackItr )
{
   // Ask for the pion fit
   FAItem< TDKinematicFit > pionFit = (*trackItr).pionFit();
   report(INFO, kFacilityString) << "Track Momentum"
       << (*pionFit).momentum() << endl;
}
```
Creating Code: Finding Out More

*Search for Information about different types of objects here:
www.lns.cornell.edu/restricted/webtools/cleo3
*Example, look up TDKinematicFit in an “Identifier Search”. You will get a list of files where TDKinematicFit is defined.

*Note that TDKinematicFit “inherits” from KTKinematicData.

*click on KTKinematicData. click on KTKinematicData.cc. Scroll down, and you can see all the different pieces of information you can get from the pionFit.

*Example, print out the pionFit charge, (*pionFit).charge()
Creating Code: Undefined Symbols

*One common problem in writing code is undefined symbols.

*Explore undefined symbols by removing Navigation from the “CLEO3_LIBS” list in the Makefile file.

*See if it compiles (do a “gmake clean” in build area first).

*You’ll get the following error when you try load it in suez

prompt> suez -q
Suez> proc sel MyFirstProc
   %ERROR-DynamicLoader.DLSharedObjectHandler: /home/rcg24/cleo101_test/build/20050316_Fhlib/MyFirstProc.so_20050316_FULL: undefined symbol: _ZNK7TDTrack11pionQualityEv

*Undefined symbol errors come out mangled. De-mangle with c++filt see below:

prompt> echo _ZNK7TDTrack11pionQualityEv | c++filt
TDTrack::pionQuality() const
Checking Out Code

*Depending on your project, you may need to check out and modify existing code from the libraries.

*Unless you are working on code development, use \texttt{cleo3cvs} command.

*Always checkout the same “tag” or version of the software package as what is used in the fixed release you are interested in.

*To get the tag of a package used in a release, you may do the following.

\begin{verbatim}
example
prompt> c3rel 20050316_FULL
prompt> grep NavigationProd $C3_LIB/pkg_versions
NavigationProd Analysis v12_18_00 prc
$
\end{verbatim}

*In release 20050316\_FULL the version tag of NavigationProd is v12\_18\_00. You can use the following command to check out this version.

\begin{verbatim}
example
prompt> cleo3cvs co -r v12_18_00 NavigationProd
\end{verbatim}
Checking Out Code: Make a change

*Now, just to test things out, emacs Navigation-Prod/Class/NavTrackProxy.cc and make the following change.

original code

// constructors and destructor
//
NavTrackProxy::NavTrackProxy( NavigationProd* theCallingProducer)
   : m_ptrTable( new value_type ),
     m_masterBlock( 0 ),
     m_navigationProd( theCallingProducer )
{
}

Modified code

// constructors and destructor
//
NavTrackProxy::NavTrackProxy( NavigationProd* theCallingProducer)
   : m_ptrTable( new value_type ),
     m_masterBlock( 0 ),
     m_navigationProd( theCallingProducer )
{
     report(INFO,kFacilityString)<<"I am in my NavTrackProxy"<<endl;
}

*We simply added a print statement. When the NavTrackProxy class is initialized, the message “I am in my NavTrackProxy” will be written to the screen.
Checking Out Code: Test the Change

*Now compile your NavigationProd.

*Change MyFirstProc.tcl to only go over 1 event instead of 1000.
*Run MyFirstProc.tcl in suez.

*Check output of “prod lss” (NavigationProd should be your copy)

*Part of output for event 1 should be:

```
>> Mon Jun  6 17:07:57 2005 Run: 200978 Event: 1 Stop: event <<
%% INFO-NavigationProd.NavTrackProxy: I am in my NavTrackProxy
%% INFO-C3ccProd.CcShowerAttributesProxy: Running with CC run-to-run corrections
%% INFO-C3ccProd.CcShowerAttributesProxy: Running with CC hot list suppression availabl
```