An useful ATLAS top analysis package: BoostedRealm

Index

Introduction

The BoostedRealm framework is designed to perform a full analysis, from D3PD to fynal plots, for the top and ttbar differential cross section with ATLAS data in a self-cosistent and guided environment. This is also a good starting point for other top analysis, in fact it is enought flexible to be integrated with other pieces of code; some bonus codes already in and lots of typical analysis issiues also ready.

This framework's core is constituted by 4 main codes described below and a series of scripts to easy exec programs and for different analysis pourposes, also described in following sections.

BoostedRealm is a **TopRootCore(TRC)** based framework that garantee a quite up to date and reliable environment. If you are new to TopRootCore you can have look at the TRC Twiki. The framework uses an official TRC package called **TopD3PDBoosted** developed by Avi Gershon for a semi-leptonic boosted and resolved top event processing(D3PD2MiniSLBoosted). The **D3PD2MiniSLBoosted** code to transform D3PD to ntuple appling proper corrections and cuts is part of this package. A local and GRID submission script is provided.

The **HistoMakerMaster** code, to produce histograms starting from the ntuples and to reconstruct physics objects, is also integrated in the TopD3PDBoosted package. This code also performs the particle level selection (thanks to Ana Ovcharova for her contribution) and a boosted tagging selection using the **Top Template Overlap Method**.

The final analysis step is made by 2 codes:

- the one to produce **data/MC** and other control plots.
- the program for the **unfolding** and cross section evaluation.

Both these codes recognaize the input files and histograms typology by names in order to speed up and automatize the elaboration and future modifies; apply luminosity correction; rebin and set histograms in editorial format; evaluate statistic and systematic errors; are customizable via parameters file.

Most of the instruction are given when you run the code, so pay attention and read them.

D3PD2MiniSLBoosted

The D3PD2MiniSLBoosted code processes D3PD into TRC ntuple format called MiniSL. TRC is a quite complicated infastructure, so I'll describe just the most useful classes for a typical analysis user scope (read also the *General Conventions > Infrastructure* section to know where to find programs):

- **SemileptonicBoostedSelection.cxx(.h)** is where the cuts are defined.
- **MiniSLBoosted.cxx(.h)** and **MiniSLOrig.cxx(.h)** are the the program that fill the the output ntuple branches.
- **AppBaseBoosted.cxx(.h)**
- **MiniSLRunBoosted.cxx(.h)**

Some useful parameters described in the AppBaseBoosted.cxx and AppBase.cxx files and already applied by the the sublission scripts:
-boost for boosted analysis.

-noTrim for no trimmed large R jets and to save all events at cut11
-useClusters to save cluster info
-sysOn to produce systematics only.
-useLoose to produce also loose data.
-useTruthParticles to enable branches at truth particle level.
-mcTruth_noCut to save events just after the baseline cuts only(only truth variable saved).
-allTrimFatJets to save all trimmed fat jet and not only the one passing the corresponding cut.
-mcType fullSim if you are running on MC data
-dataStream Muons if you are running on data muon stream
-dataStream Egamma if you are running on data electron stream

HistoMakerMaster

HistoMakerMaster is responsable for the histo production, the event reconstruction, the template overlap selection and the particle level selection. This is a TRC integrated program structured in 3 main functions: initialize(), execute() and finalize() inerited. Read the *General Conventions > Infrastructure* section to know where to find programs.
Histograms are managed by the **SmartH** class in the namesake files. It recognize the histo name convention (so change here in case on name convention changes) and automatically set binning, title, axies and the other histo properties. To add a new histo:

- you need to declare a *SmartH* object in *initialize()*. It will be added automatically in a map ordered by the same name of the histogram
- call the *Fill1D(...)* [or *Fill2D(...)*] class method in the *Fillall()* function giving the 4vector of the object to be plotted(hadronic top), the variable you want (pt) and the event scale factor.

The physics objects are managed by the **SmartVector** class in the namesake file. This is an overloading of the root TLorentzVector class with some more features; the most useful is the *IsFull()* class method to check if the 4vector is ever been filled (useful for the global program variable managment). To get the original TLorentzVector object call the *Vector()* class method.

In the *execute()* method it open the input ntuple (all charged in *initialize()*) and starts the loop over the TTree calling *executeEvent()*. Here you can add all the operation you need to do for every event like cuts and variable filling. In *execute()* input file properties are recognized, the normalization factor (in *MC_Normalize()*) and the event weight (namesake variable) are evaluated, the physic objects are reconstructed. The Particle Selection is done in *Recluster()*.

This program can also perform the **Top Template Overlap method** to tag top fat jets. You need to set the proper parameter and the input template file list. Histograms with reconstructed objects selected with boosted top fat jet are seved with the tag *boostOv*. The histograms with the tag *ovValue* contain the information of the selected best matching template: the overlap value and the kinematic quantities. The algorithm is implemented in the `matching.hh` file, while the `OverlapMaster.h` file manage the tagging procedure.

The possible input parameters are:

- **-local** if you run locally with a file with a different convention for file name and path so you need to define by parameter some setting otherwise automatic. To see how to define other local parameters take a look in `LocalHistoMakerBo.sh`. **This procedure is to be used just in emergency and test case, not for ther full analysis!**
- **-f** input file list
- **-o** output histo file name
- **-onlyParticle** to save only *particle level* information and not parton level ones.
- **-particleCut** apply particle level cut. It's however done just for signal ttbar files.
- **-topAntitop** save top and antitop ture information.
- **-noCut** for signal ttbar no cut nutuple processing.
- **-estimateFakes** for data drive fake estimation. To be used over loose samples.
- **-n [nEvent]** number of events to process.
- **-ovOn** activate the template overlap method and pass the template file list. **tested just for one template file**
- **-batchNum** number of batch job got fron the number in the file list
- **-selectedEventLog** to create a list of runNumbers and eventNumbers of events passes particle or reco selection.

ShinyPlot

The main class for this feature is ShinyPlotter (in `$SHINY_REALM/ShinyPlotter.h` file). It takes all the histograms produced and merged recognaizing them by the name and automatically sets some parameters (title, binning, color, ...) always by their name. To add new naming options go to `$SHINY_REALM/ShinyPlots.h > HistoSetting()`.
The code produces a plot of every histogram in the input root file and a stack to compare real-data with the sum of the signal and bkg contributon. It is pre-implemented to create rate and efficiency plots matching automatically the needed pair of histos; for example it makes the efficiency reco/particleLevel for all the kinematic quantities in input looping over all the possible reco and particleLevel histos matching the correct couple. It is easy to add similar plot based on the same principle, in this way you do not need to write the same procedure for every histo.
The program produce also the normalized responce matrices.

Setting and printing are demanded to ShinyPlot class (in `$SHINY_REALM/ShinyPlot.h` file) and a map of this objects is created in *ShinyPlotter*. If you want to add or change histograms setting you need to look into `$SHINY_REALM/ShinyPlot.h` file.
If **-s** parameter is enabled the systematics are evaluated. A systematic error table is also produced in this case.
Files are also renormalized by the correct luminosity defined in `$SHINY_REALM/ShinyPlotter.h` as `#define LUMIDATA ...`. Check luminosity value for your particular sample. (see *Miscellaneous* chapter)

All the plots are saved in png, eps and root format in the `$PLOT_REALM/last` dir.

- By default it takes the MC qcd bkg. To use the data driven bkg set the variable **ddQcdOn** equal true in `$SHINY_REALM/ShinyPlotter.h`. It's hardcoded because it is a temporary solution for testing.

UnfoldMaster

The program performs the unfolding for a single kinematic quantity per time (i.e. hadronic top pt) and produce the corresponding differential cross section distridution; different cross sections can be performed (relative, absolute, ...). Also different unfolding methods are available (MatrixInversion, SVD-Tikonov, ...). Kinematic quantities, unfolding method and cross section typology can be setted via parameter file.
The .cxx files are in `$UNFOLDING_REALM/src` while .h files are in `$UNFOLDING_REALM/include`.

Global parameters and variables are defined and initialized in the `globals.cxx(.h)` file(s).
The main class is **UnfoldMaster** that manage all the processing using a map of **UnfoldBringer** objects, one for each input sistribution (systematic, data, bkg, signal, ...).
The cross section evaluation is done by the **CrossUnter** class.
The systematic error evaluation is done by the **SysCrafter** class. For now the systematic errors are summed in quadrature.
The unfolding is done by the **UnfoldSmith** class using the RooUnfold package (introduction page and documentation).
The `MaicSkills.cxx(.h)` contains some general and useful standalone functions.

All plots are saved in `$PLOT_REALM/unfoldLast`.

As input files you will need the *standard* (or *nominal*) file _with data, signal ttbar and bkgs_ and the *noCut* (or *generated*) file. You can add the *loose* (*fake*) file for data-driven fakes bkg and all the generated systematic files, if you want. All files must be in the list file as generated by `UnitedHisto.sh` (see below).
Parameters will allow you to enable data-driven fakes bkg, systematics and closure test (correct files needed).

program description

The program creates a map of **UnfoldBringer** objects named with the last 2 tags of the corresponding input file, i.e. using file *el_fynalStandard15fbAllOk_nominal_single.root* the map name will be *nominal_single*. Some other map elements are created for special systematics(see `$UNFOLDING_REALM/include/SpecialSys.sh`). An other identical map is created and will be rebinned with only one bin to evaluate the **integrated cross section** value. For all map elements the program performs (in this order) rebinning, bkg subtraction, unfolding and cross section evaluation. Then the statistic and systematic errors are evaluated and at last plots and systematic tables are printed in `$PLOT_REALM/unfoldLast`.

To debug and plot unfolded number of events instead of xsec go into `UnfoldBringer.cxx > GetCross( string cx )`. Plot title and view setting are defined in `FinalPainter.cxx`

General Conventions

Programs are managed by some parameter files that will be opened for editing every time you exec the corresponding script. You can comment a line in the parameter file typing # at the begin.

Infrastucture

The main dir is `$MASTER_REALM`. From here you can exec **all** the programs via bash scripts, so go here every time you need to exec a *.sh file. File are usually been executed this way *./program.sh*. If you will create a new one remember to do `chmod 744 -R program.sh` to gain the correct user rights.

trc

Each TRC package in structured in subdirectories:

- **Root** contains *.cxx codes
- **"packageName"** contains headers *.h codes
- **util** contains *.cxx codes to be executed directly
- **run** contains link to other packages and running files. In *TopD3PDBoosted* case you can find here default produced ntuple and grid zipped file.
- **script** in *TopD3PDBoosted* case here you can find useful script like the one for grid submission.

Nameing

The whole infrastructure require a quite precise nameing convention for both files and histograms. This allows to recognize the type of file (systematic or nominal or generated or ...), the type of the histogram and to treat them automatically using, in example, the same setting coherently (like binning, colours, ...).

Tar file repository (not to be used any more)

- When you produce the ntuple from D3PD (or them skimmed files) you must give a ID number for the processing. All tar files produced should be placed in a directory with that ID in the name(if you are doing local production using batch submission script they will be automatically saved in the right place). Example: if ID=666 then the folder name will be 666__[date of production]_production.

Ntuple folder stucture and name

- When you produce the ntuple from D3PD skimmed files you must give an ID number for the processing. All files produced will be placed in a directory with that ID in the name. The local production using batch submission script will automatically save them in the right place, i.e. in `$BOOSTEDCOMMONDIR/NtupleProduction/ID_[date of production]_BoostedNtupleDir`. The **data12_8TeV** sub-folder will be created here if they are data ntumple: in case of MC ntuple **mc12_8TeV** sub-folder will be created. In the MC ntuple case another sub-folder level will be created for each MC type: ttbarMcAtNlo, ttbarPowhegPythia, wjets, zjets, ecc.
  The next folder level will represent the GRID dataset name while the next one the origial skim file name. Here you will find at last the ntuple files: **el.root** and **mu.root** for the electron and muon channel respecively. For loose and systematic production there is a tag in the ntuple name like **el_loose.root**.

Folder Level	**example**
Repository dir	$BOOSTEDCOMMONDIR/NtupleProduction/666_20131005_BoostedNtupleDir

data/MC	**mc12_8TeV** or **data12_8TeV**
MC type **(only for MC)**	ttbarPowhegPythia
dataset	user.matteo.data12_8TeV.periodA.physics _Egamma.PhysCont.NTUP_TOPBoosted_SLEL.grp15_v01_p1400_p1401_86_data.130926012715
skim file	user.matteo.015316._00111.XYZ.root.tgz
ntuple	el.root

Histogram name

The histogram name should be composed by 5 or 6 specific tags separeted by underscores. If you want to add a new option in any tag, you will need to modify the SmartH class, unfolding and the plotter programs tolet them recognize the new tag. The name structure follows:

tag1, : the event level of the object plotted in the histogram like reconstructed or truth object. The possible options up to now are:

`reco`: for reconstructed quantities
`true3`: for truth quantities at parton level
`particle`: for truth quantities at particle level
`fake`: for data-driven fake leptons quantities

In case of 2D histograms the same 1D histogram level will be reported for both the quantities plotted separed by VS like particleVSreco

tag2, : the object the histogram refers to like hadronic Top or W boson. The most common options up to now are:

`hadTop`: for the hadronic top
`lepTop`: for the leptonic top
`diTop`: for the ttbar system
`lep`: for the lepton
ecc....

tag3, [optional]: to recognize some plot representing the same distribution but differing for something. For example the reco ttbar system could be reconstructed in our case using trimmed jet with d12 selection or the template overlap method; different tags have been used.

`boostOv`: if template overlap method is used in reconstructed object
`boostTrim`: if trimmed jet d12 selection is used in reconstructed object
`leadingPt`: if the truth particle level top is choosen as the leadind pt truth AntiKt10 jet
`leadingPt`: if the truth particle level top is choosen as the truth AntiKt10 jet best matching with a truth hadronic top

tag4, : recognize the histogram type (1D or 2D). The only possible options up to now are:

`dist`: for 1D histos
`respo`: for 2D histos

tag5, : is the quantities plotted in the histogram. The most common options up to now are:

`pt`: transverse momentum
`m`: mass
`eta`: pseudorapidity
ecc...

tag6, : the data type on the events, if it is a MC or real data event, form which kind of MC it came from. The only possible options up to now are:

`data`: real data
`ttbarPow`: signal ttbar PowHeg MC sample
`ttbar`: signal ttbar McAtNlo MC sample
`diboson`: diboson background MC sample
`stop`: single top background MC sample
`qcd`: qcd background MC sample or data driven one
`wjetsWhh`: w+jets background MC sample
`wjetsWc`: w+jets background MC sample
`wjetsll`: w+jets background MC sample
`zjets`: z boson + jets background MC sample
`zprime`: z prime boson MC sample

Example of an histogram name: `reco_hadTop_boostTrim_dist_pt_ttbarPow`

Installation

download form SVN the following package:

```
svn co svn+ssh://matteo@svn.cern.ch/reps/atlas-matteo/matteo/tags/BoostedRealm-01-00-00
BoostedRealm
```

and launch the installer

```
cd BoostedRealm

source boostedRealm.inst
```

use this tag TRC: 14-00-15 When it'll ask you to change tha pakage file, go to TopD3PDBoosted line and substitute it with

```
atlasoff/PhysicsAnalysis/TopPhys/TopD3PDBoosted/tags/TopD3PDBoosted-13-01-01
```

Launch

`./AdaptToGrid.sh` (to set the last version of the histo maker programs)
`./FastJetCompile.sh`
`./BuildTRC.sh`

Every time you enter your system you need to setup the environment:

```
realm_setup
```

TopRootCore Installation only

You can re-install only a new TRC version like this

```
cd $INSTALL_REALM

source trc_installer.sh
```

!! ATTENTION !! When you get *TopD3PDBoosted* from SVN and you want to save events before the last cut, you have to comment the following line in `$TOPD3PDBOOSTED_HOME/Root/MiniSLRunBoosted.cxx` (around line 170):

```
if (m_boosted_run) topReco.doBoostedReco(thisEd,slbase); else topReco.doResolvedReco(thisEd,slbase);
```

Important remark: the local analisys needs the *TtjetsClassifier* package, not in the official TRC version, and FastJet. In order to use these 2 packages the *Makefile* should be changed. Some other programs too. For that reason in the official *TopD3PDBoosted* SVN repository none of those modifies or packages are present. In `$REALMPATH/moveFast` those packages has been instead saved. You can find the *TopD3PDBoosted-MakeFile* and the *HistMakerMaster* for both cases (with and without particle selection packages). You can update this part using:

```
./AdaptToGrid.sh
```

and follow instruction. **Exec this script after every installation is strongly recomanded.** After that you would probably need to compile with the clean option to clean some libs. Not always needed.
It you run on skimming you ALWAYS want to run TRC locally !!!

A Tipical Analysis Procedure

Setup BoosetdRealm enviornment, if not already done, using command

```
realm_setup
```

and go to the master dir

```
cd $MASTER_REALM
```

When you exec a script most of the info will be given you during the execution on the screen so READ CAREFULLY the instruction given

Make sure to have the right TRC version and the TopD3PDBoosted version working locally: about that see also the Install paragraph on top. To have locally working version use

```
./AdaptToGrid.sh
```

with the l option and follow the instructions. It copy the correct Makefile and HistoMakerMaster.cxx and move the FastJet and the TtjetsClassifier package. This program can be also used, with a certain care, in order to check which version have you got now and to update the programs that are substituted in the procedure.

Compile TRC

In order to compile TopRootCore(TRC) exec

```
./BuildTRC.sh
```

You can set some addictional options:

- `./BuildTRC.sh clean` to clean up all libraries
- `./BuildTRC.sh findPack` to launch the find_packages.sh tool
- `./BuildTRC.sh buildAll` to lanch build-all.sh instead of compile

(FastJet enviornment will be also settedup but you will need do it again before to launch any TRC program, if you are using FastJet of course. It is already done is you use the following scripts).

Skimming

... Ask Alberto Mengarelli or Matteo Negrini for now ...

Ntuple production from D3PD

You will now produce your ntuple locally from skimmed files. First of all you need to have on your local machine the skimmed files. If you have not than you have to download from the GRID. Use dq2-get command (if so it would be better to save them in `$BOOSTEDCOMMONDIR/SkimProduction`) or subscription procedure from Panda webpage. See Miscellaneous chapter for help.

Make skim list

Produce the list of skim files. If you used the subscription to get skim so you need to have the list of skims on the GRID. If you do not have yet do:

(setup dq2)

```
dq2-ls user.mnegrini.data12_8TeV.*SKIM_0_Fjet150_Trigger/ > skims_datasets_data.txt
dq2-ls user.mnegrini.mc12_8TeV.*SKIM_0_Fjet150_Trigger/ > skims_datasets_mc.txt
```

Now create the list using

```
./MakeSkimList.sh
```

If you used dq2-get or a simple copy do

```
./MakeSkimListNotSubsribed.sh
```

All the list files will be saved in `$LIST_REALM/skimFullSamples`.

Divide skim list

In order to exec ntuple production in parallel batch jobs you need to split the ntuple list file than exec

```
./DivideSkim.sh
```

All the list files will be saved in `$LIST_REALM/skimDivided`. To decide how to divide lists look into the .sh file.

Ntuple production

This operation will be done by the program`$TOPD3PDBOOSTED_HOME/util/D3PD2MiniSLBoost.cxx`. For more information look at the Introduction chapter.

To test and run locally the code use the following script. It's not a "stable" script for its intrinsic test nature so take a look at it before to launch. Exec

```
./LocalNtupleMaker.sh
```

In order to launch the production (also small samples) in batch exec

```
QsubNtupleMaker.sh
```

to not overwrite the previouse files you have to decide the **-n** parameter. This will identify this ntuple production version from now on. It identify the production number and it must be an increasing number from 001 on, take a look to past production numbers to not overide.
The **-x** parameter allows you to process differnt samples.

Annoying operations

Now you have to prepare ntuples to be analysed in some annoying but neccessarily step.

Make Ntuple List

Produce the lists of ntuples to be analysed divided by type of data(real-data, ttbar signal, w+jets, ecc..). They will be saved in `$LIST_REALM/fullSamples`.

```
./MakeNtupleList.sh
```

you can select which list to do with the **-c** parameter.

Divide List Files

Two scripts to divide the just created lists of ntuple. The first one divide the list of all real data per data-Period. The other divide all the lists in smaller files to be easily launched in batch locally. Exec in that order

```
./DivideDataPerPeriods.sh
./DivideLists.sh
```

To decide the number of ntuples per list you should open `DivideLists.sh` and change the value of the variable *every* in the proper *if* condition, in the peace of code just after this line:

```
# decide how many file per fileList
```

Histogram Production

This part of the analysis is responsable of histogram production and object reconstruction, as well as some further event treatement like normalization and file mergeing.

Normalization and Check

In order to normalize the events to the MC production luminosity every single events is divided by the the production luminosity characteristic of this sample => `luminosity = xSec * kfactor / nEventsGenerated`. To retrieve the nEventsGenerated the programs look in the files `$MANUSCRIPT_REALM/el_FileListNevents.txt` and `$MANUSCRIPT_REALM/mu_FileListNevents.txt`. If you need a sample not in this file you need to add by hand.

TO DO: link to ami to retrieve production numbers **TO DO**: generate the file from a script

Histograms

The program that produce histograms is `$TOPD3PDBOOSTED_HOME/util/HistmakerMaster.cxx`, take a look in the Introduction chapter.
You can launch it locally executing (It's not a "stable" script for its intrinsic test porpose so take a look before to launch.)

```
./LocalHistMakerBo.sh
```

To launch the complete (or partially) analysis in batch exec

```
./QsubHistoMakerBo.sh
```

setting the proper parameters. To set the HistoMakerMaster.cxx write them down in **-p** parameter. For possible parameters look in Introduction chapter.

Merge histograms

Now you need to add together the produced histogram files. Exec

```
./UnitedHistos.sh
```

You can control the adding by some self-explaining parameters. We discuss here only the **-x** parameter that allows different mearging group. Different options are avaible:

- **standard** mearge together signal ttbar PowHeg, background and real-data files.
- **loose** mearge data-driven fake lepton bkg files.
- **sys** mearge all systematic files
- **nocut** mearge signal noo cut ttbar files
- **closure** produce 2 half-signal ttbar files for the closure test
- **all** produce all the above
- **custom** you can choose which file to mearge commenting/uncommenting lines for a text file

And the **-o** parameter that allows you to give a specific string in the merged file name so you can easily recognize with grep, ls or similar methods. A file with the list of all the merged file with the same string specified in the **-o** parameter will be created for each channel.

Data/MC comparison

In **-i** put the file list name produced by `./UnitedHistos.sh`.
The **-c** parameter also allows you to create a duplicate of the plot production with present date and time.
The **-s** parameter enable the systematic evaluation (if present in the input list).
For a description of the program read the *Introduction* chapter. To compile the data/MC comparison do

```
./PlotCompiler.sh
```

and to launch

```
./PlotLauncher.sh
```

with the correct parameters.

Unfolding

For a description of the program read the Introduction chapter. To compile and launch the unfolding program do

```
./CastUnfold.sh
```

with the correct parameters. The input file list should be the one produced during the mergeing step. After complile it will ask you to check if everything is ok and if to exec the program.

Extras

List of events passing cuts

Fist of all you have to run QsubHistoMakerBo.sh with the `-selectedEventLog` parameter. Then lanch

```
mergeSelectedEventFiles.sh
```

to merge the single lists produced by each jobs. The complete lists will be in `$LIST_REALM/selectedEvents` and contains the run number and the event number for all events passing the selection, both for particle or reconstruction one.

Miscellaneous

LumiCalc and GRL

Download the GRL you used for the data production from here `$TOPD3PDBOOSTED_HOME/run/data/TopGoodRunsList/2012/` then go to the lumiCalc webpage. Here upload your GRL in the *GRL XML File:* field.

In TRC you can find the GRL the program uses in `/home/ATLAS-T3/franchinim/Top/BoostedRealm/TRCs/TopRootCore12/TopGoodRunsList/data/2012/` (check in `$RUN_REALM/control/setting.txt`).

Leave other fields as by default. If you need to calculate the luminosity of a subsample of your GRL add the option `-r firstRunNum-LastRunNum` as explained in the lumiCalc web page a little below.
In order to retrieve the period run number go to the ATLAS Run Query and follow the examples. In example, to have runs of periodG 2012 type *f r data12_8TeV.periodG / sh ready*.

For more info about LumiCalc go to LumiCalc Twiki.

Useful link for top GRLs.

Top Group MC12 Common Conventions also has the uptodate top and ttbar MC values.

Download Files from the GRID

Very very delicate procedure. Subscription way has been tested only on the Bologna Tier3 and gives some problems. Try using dq2. To make it's "setup" go to this Twiki or do:

```
setupATLAS
localSetupDQ2Client
voms-proxy-init --voms=atlas
```

then do something like

```
dq2-get -a -f "*.root* user.[cernuser].*_[productionNumber]_[option]*"
```

Make downloaded files visible to other user wuold be nice so type

```
chmod -R 777 [filesWithWildCard]
```

Some useful dq2 tips:

- -c download only completed datasets.
- -V
- dq2-ls -r [datasets] gives a summary of the job status

Note

- How to define a custom pallette in `$SHINY_REALM/ShinyPlotter.h -> CreatePalette()`.
- A tokenize function in `$SHINY_REALM/ShinyPlots.h -> Tokenize()`
- In *MagicSkills* class a function to remove white spaces from string is implemented: **RemoveSpace(** string s **)**.