UC SANTA CRUZ

ATLAS EXPERIMENT

CERN

ATLAS Induction Week

Giordon Stark
Oct 24th, 2019
giordonstark.com

USING FOR ANALYSIS CODE GITLAB

indico://event/831761/

Overview of Today

Best practices for managing code in GitLab

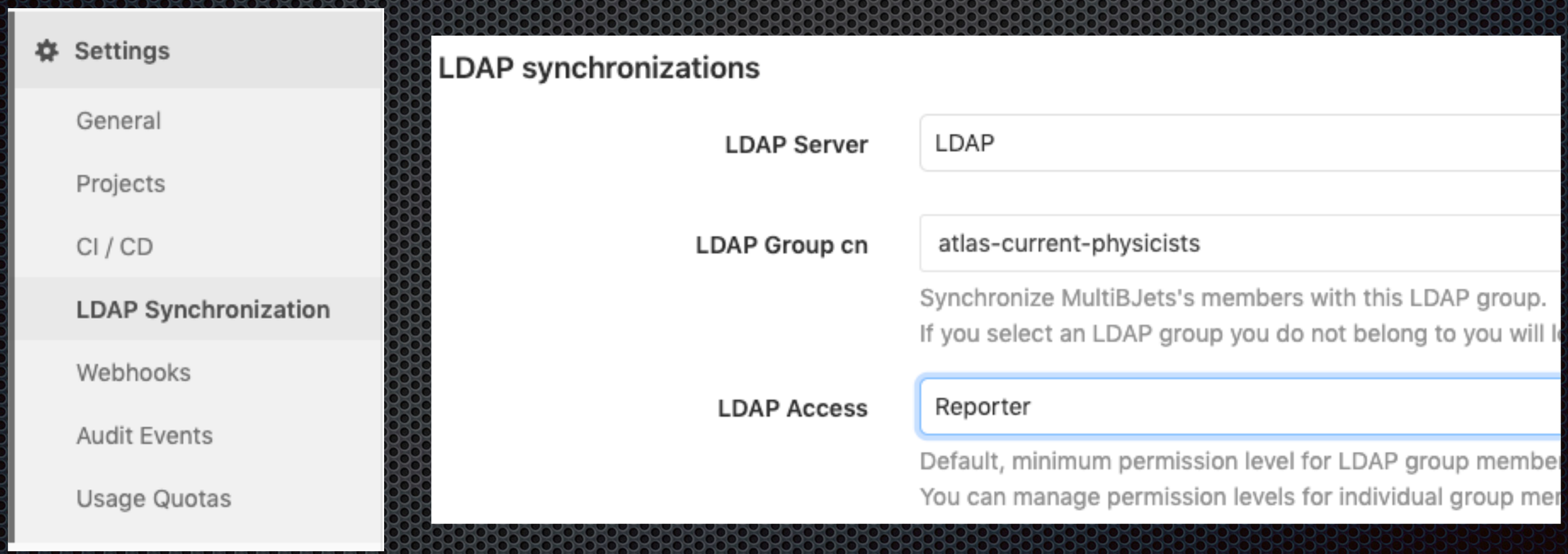
GitHub for expert developers is ok

- Repository layout in Release 21
- Using submodules and access tokens
- Setting up Continuous Integration for testing and documentation
  - see my talk with Mario Lassnig at S&C workshop on documentation best practices
- Building and deploying docker images

Full ASG support for all Analysis code in GitLab

Making a group?

- add `atlas-current-physicists` as a reporter using LDAP synchronization



- all repos in the group will be readable by ATLAS members

Want a self-guided lesson?

- Check out the CI/CD session I led at the USATLAS 2019 SW Bootcamp at LBNL: https://kratsg.github.io/2019-08-19-usatlas-computing-bootcamp/

Schedule

	Setup	Make sure gitlab.cern.ch works for you, now!
00:00	1. Introduction	What is continuous integration / continuous deployment?
00:05	2. Exit (light) Codes	What is an exit code?
00:25	3. Being Assertive	What happens with assertions in python?
00:35	4. Understanding Yet Another Markup Language	What is YAML?
00:40	5. YAML and CI	What is the CI specification?
00:45	6. Coffee break!	Get up, stretch out, take a short break.
01:00	7. Hello CI World	How do I run a simple CI job?
01:15	8. Handling Your Clones	How are repositories cloned?
01:30	9. Adding CI to Your Existing Code	I have code already in GitLab, how can I add CI to it?
01:45	10. Coffee break!	Get up, stretch out, take a short break.
02:00	11. Eins Zwei DRY	How can we make job templates?
02:15	12. All the World's a Stage	How do you make some jobs run after other jobs?
02:25	13. Run Analysis in CI/CD	How can I run my code in the CI/CD?
02:50	14. Getting into the Spy Game	How can I give my CI job private information?
03:05	15. Let's Actually Make A Test (For Real)	I'm out of questions. I've been here too long. Mr. Stark, I don't feel too good.
03:35	Finish	

What is continuous integration?

Automatically and continuously integrate changes to source code through compilation, style checking, unit testing, integration testing, and deployments

Join the Mattermost Channel! https://mattermost.web.cern.ch/signup_user_complete/?id=txz6dop6zjnqmcorqjw7orxx8w

What is continuous integration?

- **Continuous Integration provides a lot of benefits such as**
  - checking that the **code can actually compile**
  - verifying that the code runs fine on a sample root file
  - test the output of running the code and **alerting you to changes in cutflow or physics**
  - **compiling PDFs** for supporting notes, CONF notes, papers, or TDRs/FDRs/etc…
  - **generating static documentation** for your analysis
  - static code analysis techniques

What is continuous integration?

- **Continuous Integration provides a lot of benefits such as**
  - allowing for **custom deployments** such as:
    - creating a docker image of the compiled analysis code and uploaded to either hub.docker.com or GitLab's docker registry
    - uploading documentation to a website or a new page
    - trigger analysis reinterpretation through RECAST
    - **[expert]** automatically submitting jobs to Panda or Condor or batch nodes
    - **[expert]** automatically updating CDS with the newly created PDF when adding a new tag
    - **[expert]** build a filterable webpage of all SUSY feynman diagrams compiled from latex to svg

Why is it important?

- One of the hardest parts of starting a new project (software, firmware, thesis writing, plotting code, etc…) is **setting up the structures and foundations**
- Consistent layouts, well-written documentation, intuitive structuring
  - all of these makes the **on-boarding process easier for newcomers**, and **easier for others to help you** when you have problems!
- **Preserving code and ideas** for years to come is crucial for **reproducibility**!
- Learn how to write code that is **friendly for humans and robots**!
- Learn how to set up a testing environment so **others can contribute** and know what you expect of the code, without having to ask you!

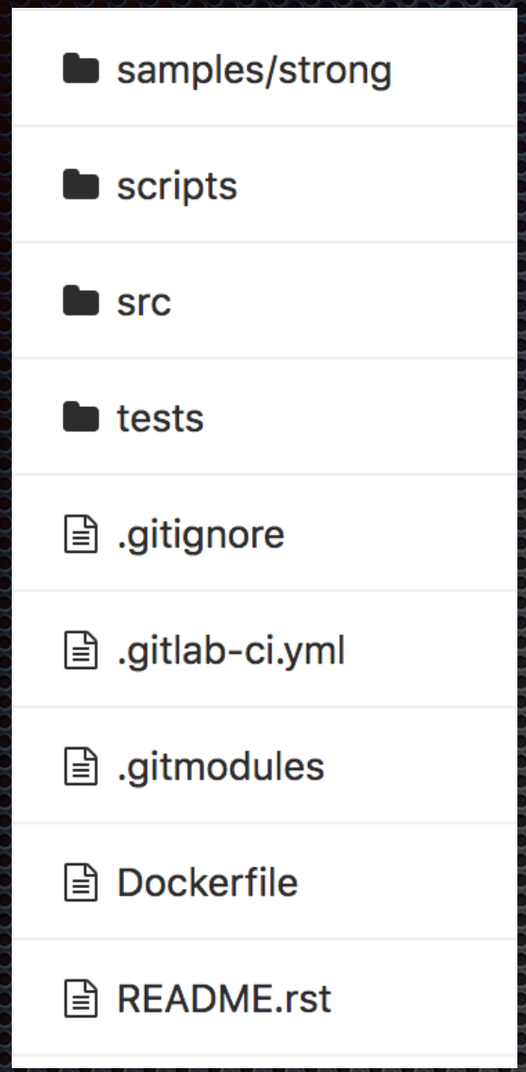
ATLAS is collaborative, your code should be too!

The Case Studies

- Today, I'll talk about this from the perspective of two codebases:
  - xAODAnaHelpers (github:UCATLAS/xAODAnaHelpers)
  - MBJ_Analysis (gitlab://MultiBJets/MBJ_Analysis/)
- xAH is an analysis successfully used by many different kinds of published analyses
  - Maintain a working master using **continuous integration** and **automatically-generated documentation**
- Multi-b-jets is the SUSY Strong production search (with two published papers) and a fast turn-around for the Release 21 scrutiny effort
  - Using continuous integration to catch changes in the **physics**
- For more, see this very nice twiki: twiki://SotwareTutorialAnalysisInGitReleases

gitlab://MultiBJets/MBJ_Analysis/

Software Layout - MBJ (I)



Top-level contains source code, tests, and miscellaneous (short) scripts

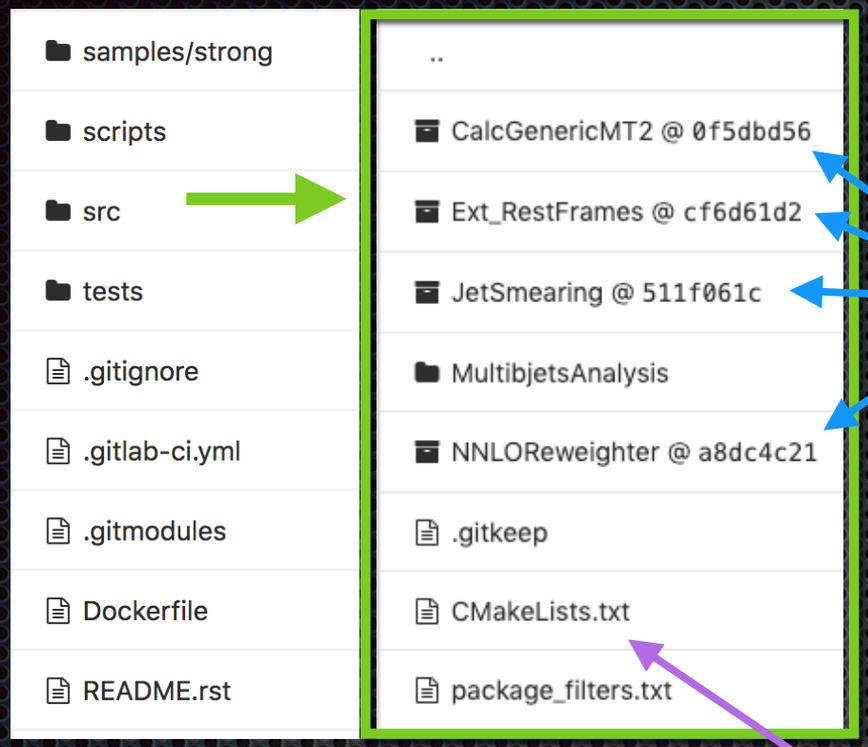
- source code uses a **submodule layout** (see next slide, `.gitmodules`)
- Add a README so users know whats up
- Take good care of your gitignore, many lazy users run `git add .` — adding all files — causes repository bloat and annoyed physicists
- `.gitlab-ci.yml` defines continuous integration (see later slides)
- `Dockerfile` — instructions on preserving code and runtime environment (more details in Lukas' talk here)

example Dockerfile

```
FROM atlas/analysisbase:21.2.87
ADD . /MBJ/
WORKDIR /MBJ/build
RUN source ~/release_setup.sh &&  \
    sudo chown -R atlas /MBJ && \
    cmake ../src && \
    make -j4
```

gitlab://MultiBJets/MBJ_Analysis/

Software Layout - MBJ (II)



Reduce the number of `git clone` a user calls manually

- **submodule layout** is a one-stop shop for your entire code
  - "**symbolic link**" to other packages and source code at a specific revision
  - grab everything with a single (recursive) git clone
  - Contains a top-level `CMakeLists.txt` which you should write (ASG recommends writing your own!)
- Use the same layout for checking out code which reduces user error and standardizes the workflow
- Top level CMakeLists.txt is committed — as it should be written for each analysis code separately (ASG recommendation)
- Use relative submodule paths where possible, as this allows cloning over kerberos, ssh, or https automatically
  - Also integrates very nicely with continuous integration

.gitmodules 405 Bytes

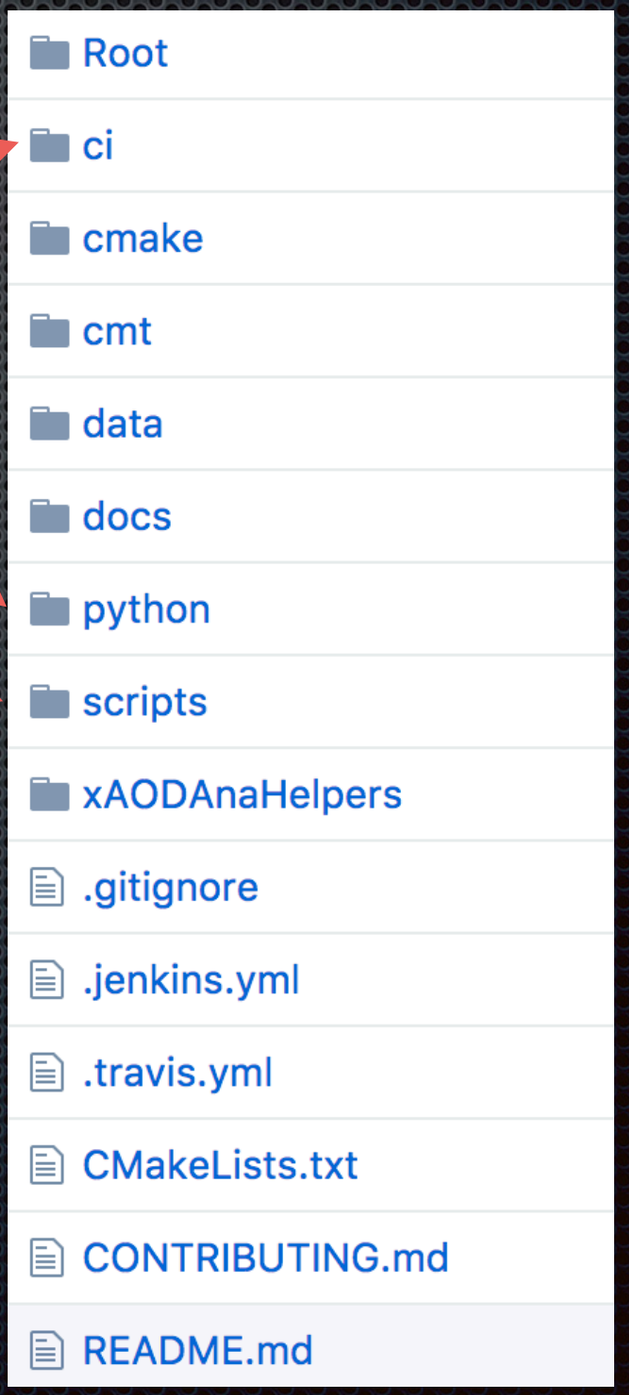
```
[submodule "src/Ext_RestFrames"]
        path = src/Ext_RestFrames
        url = https://github.com/lawrenceleejr/Ext_RestFrames
[submodule "src/JetSmearing"]
        path = src/JetSmearing
        url = ../../atlas-phys-susy-wg/JetSmearing
[submodule "src/NNLOReweighter"]
        path = src/NNLOReweighter
        url = ../NNLOReweighter
[submodule "src/CalcGenericMT2"]
        path = src/CalcGenericMT2
        url = ../../atlas-phys-susy-wg/CalcGenericMT2
```

Notice how "small" this repository is. It provides a top-level overview of the entire project!

github://UCATLAS/xAODAnaHelpers

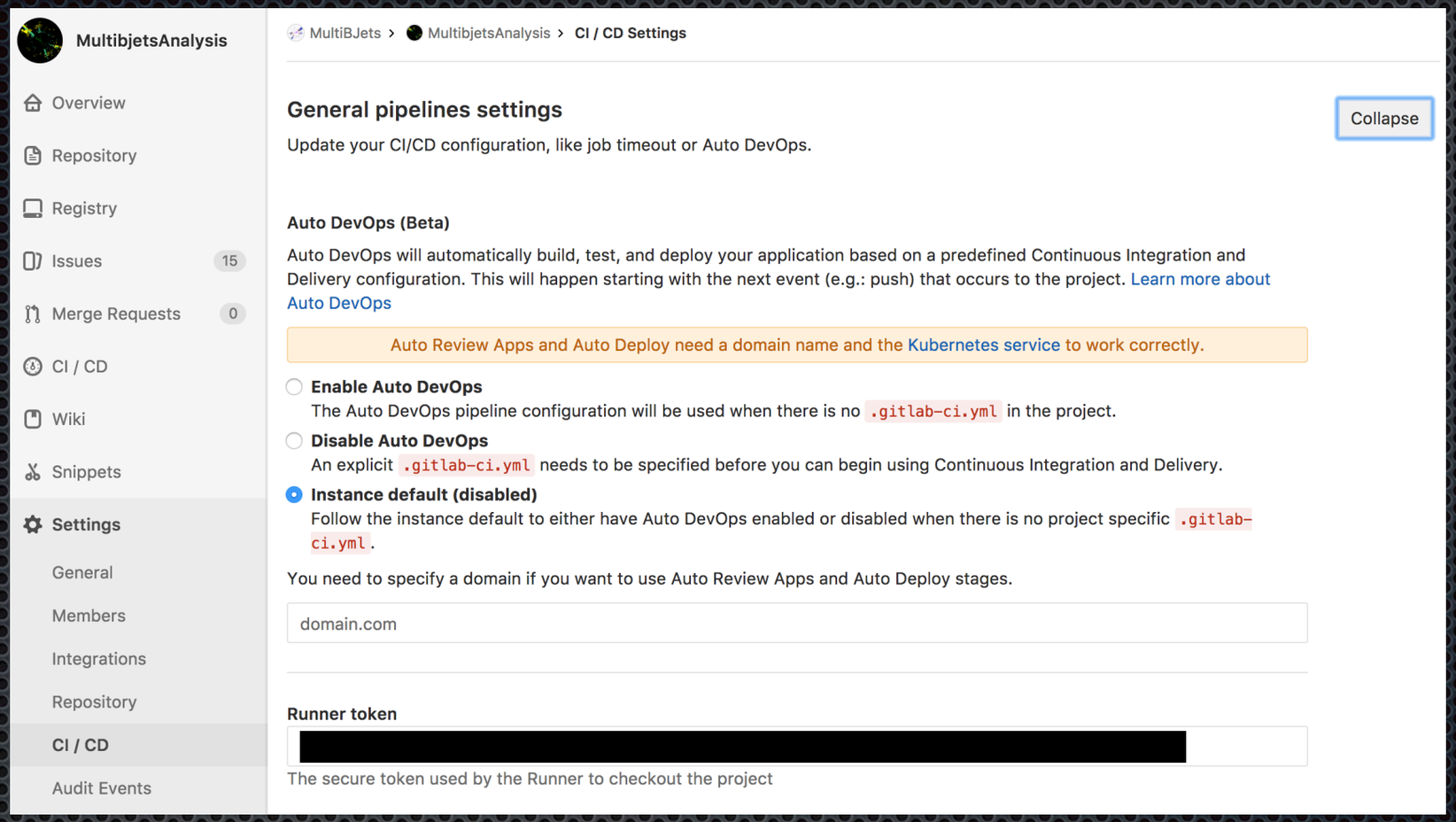
Software Layout - xAH

- xAH is a **good example** of how an analysis code or framework should be structured
  - **Separate folders** for source code, continuous integration scripts, cmake scripts, data, docs, scripts, and python packaging
- Top-level should include the `CMakeLists.txt` for your package along with a `README`
  - `.gitlab-ci.yml` and/or `.travis.yml` should be here for CI
  - CONTRIBUTING guidelines can be added as well
  - `.gitignore` to hide specific types of files (such as `build/` dir or `.pyc`)
- Some tips!
  - Your package can be **python-importable** if you put an `__init__.py` in `python/`
  - You can use `CMake` to **install other python packages** with your code (such as `NumPy` or `PyYaml`)
  - Command-line executable scripts should go in `scripts/`, **not** `python/`



https://gitlab.cern.ch/MultiBJets/MBJ_Analysis/settings/ci_cd

Access Tokens on GitLab



- GitLab uses access tokens (OAuth2 tokens) to maintain unauthenticated access on behalf of a user
- By default, an access token for a repository is a deploy token, it gives read-only access to that specific repository
- You can create one for a user by going here: https://gitlab.cern.ch/profile/personal_access_tokens and generating one
  - The access token generated will have the same level of access as the user who owns the token
  - This is useful for cloning/accessing all the submodules you might add to your project

https://gitlab.cern.ch/groups/MultiBJets/-/settings/ci_cd

Env Variables on GitLab

MultiBJets > **CI / CD Settings**

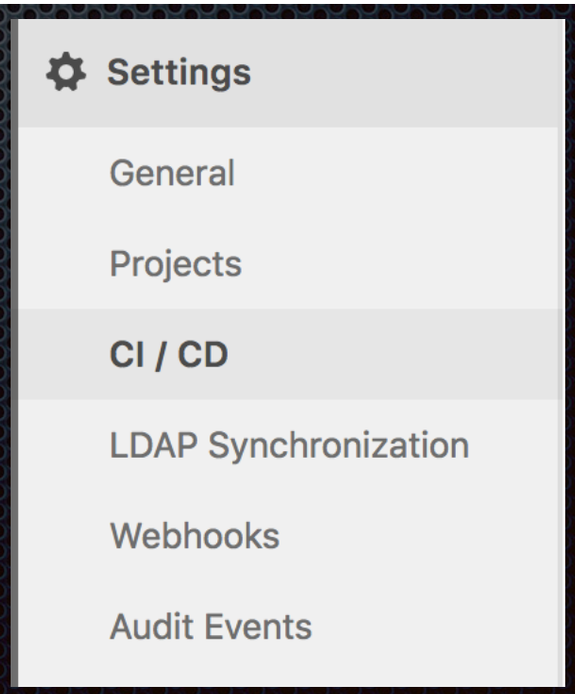
Secret variables

Variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. You can use variables for passwords, secret keys, or whatever you want.

CERN_USER	**********	Protected
SERVICE_PASS	**********	Protected
Input variable key	Input variable value	Protected

Save variables | Reveal values

- You can set up a **service account** and add either: **repository-level** secret variables or **group-level** secret environment variables with login details
  - **This is useful to give your runners access to root files over xrdcp!**
  - I suggest **group-level** if you make a **service account** for your analysis team and you have multiple repositories that would like to have continuous integration to take advantage of testing on actual root files over xrdcp
- Go to **Group** or **Repository**, click "`Settings`", then click "`CI/CD`", and you can add environment variables under "`Secret Variables`" section



"Documentation - is there an easy way?" on hn-atlas-PATHelp(at)cern.ch

Use-case of access tokens

Gordon W.

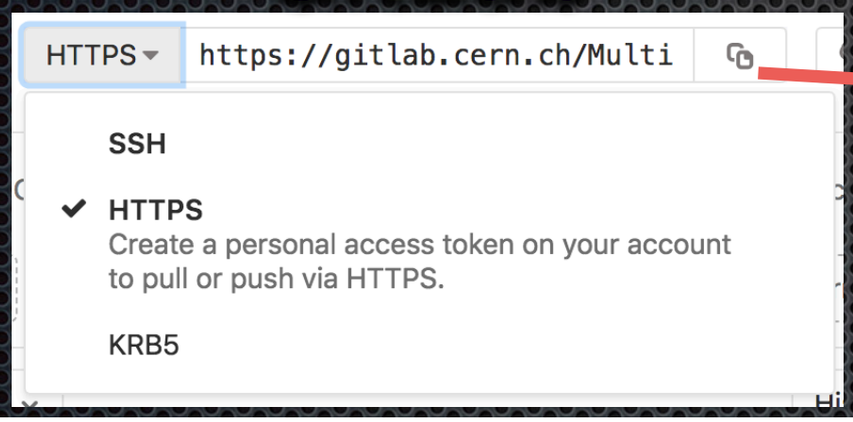
"I have an ATLAS internal project (gitlab://mathusla/TestStandOffline that I want to generate documentation with doxygen and Sphinx and host on ReadTheDocs.org (RTD). How can I do that?"

G(i)ordon S.

"You can have RTD configured with webhooks, provide it an https URL for your repository with a personal access token, and it will be enough to do everything."

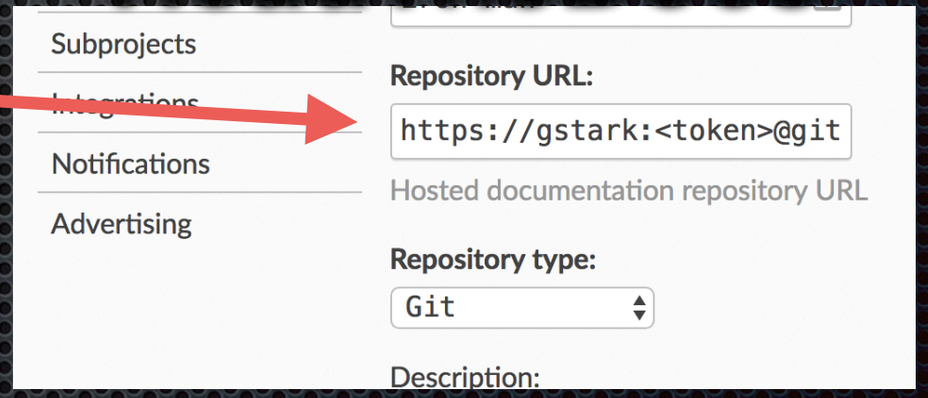
git clone https://{username}:{token}@{domain}/{group}/{repo}

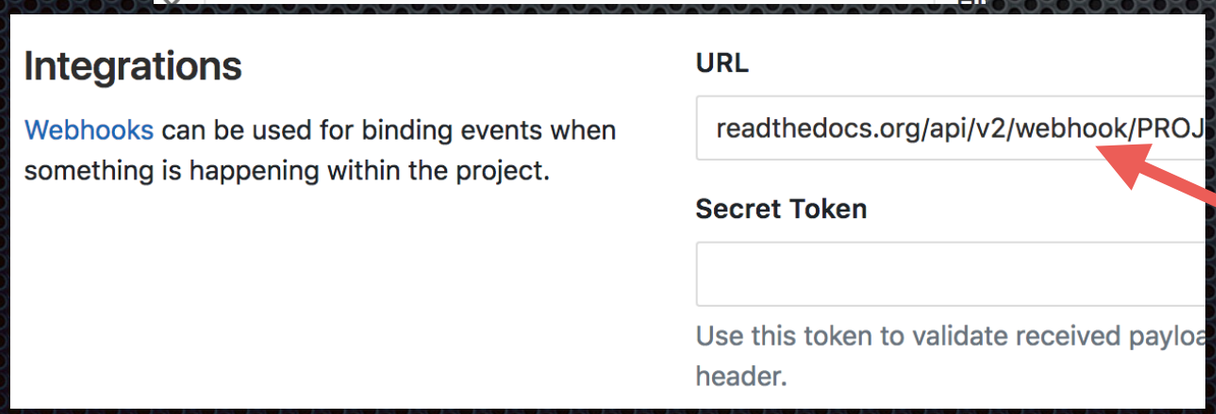
GitLab



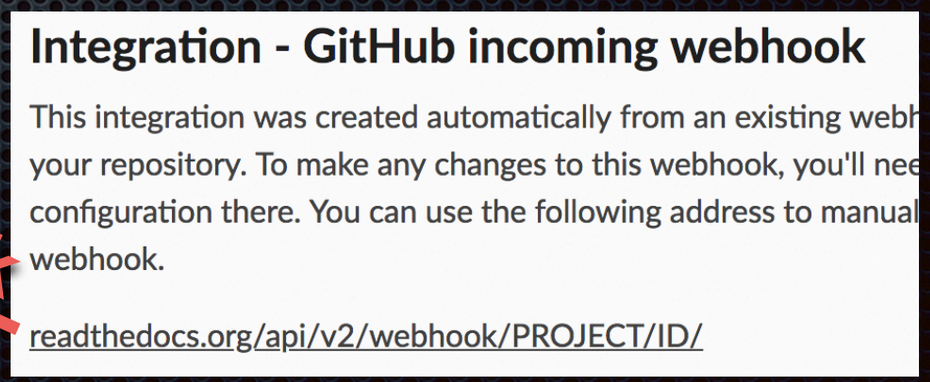
token

ReadTheDocs





webhook



Setting up CI (I)

Create a `.gitlab-ci.yml` file (or `.travis.yml` for GitHub), add it to the top-level of your repository, and you're done!

- See API reference for what you can do: gitlab-ci, travis

```yaml
script:
  - envsubst '\$ABV_CM \$TRAVIS_COMMIT \$TRAVIS_JOB_ID' < ci/Dockerfile > Dockerfile
  - docker build --build-arg ABV_CM=${ABV_CM}
                 --build-arg TRAVIS_COMMIT=${TRAVIS_COMMIT}
                 --build-arg TRAVIS_JOB_ID=${TRAVIS_JOB_ID}
                 -t ucatlas/xah:${ABV_CM}-latest .
  - docker run --rm ucatlas/xah:${ABV_CM}-latest
  - docker run --rm ucatlas/xah:${ABV_CM}-latest /bin/bash -c \
        'source xAODAnaHelpers_setup.sh; xAH_run.py -h'
deploy:
  provider: script
  skip_cleanup: true
  script: ci/deploy.sh
  on:
    branch: master
    condition: "$ALLOWFAIL = false"

 include:
    - language: python
      python: 2.7
      sudo: false
      env: JOB="Building online documentation"
      addons:
        apt:
          packages: [doxygen, doxygen-doc, doxygen-gui, doxygen-latex, graphviz]
      install: pip install -r docs/requirements.txt
      script: cd docs && doxygen && make html
```

.travis.yml

Validate it: https://lint.travis-ci.org/

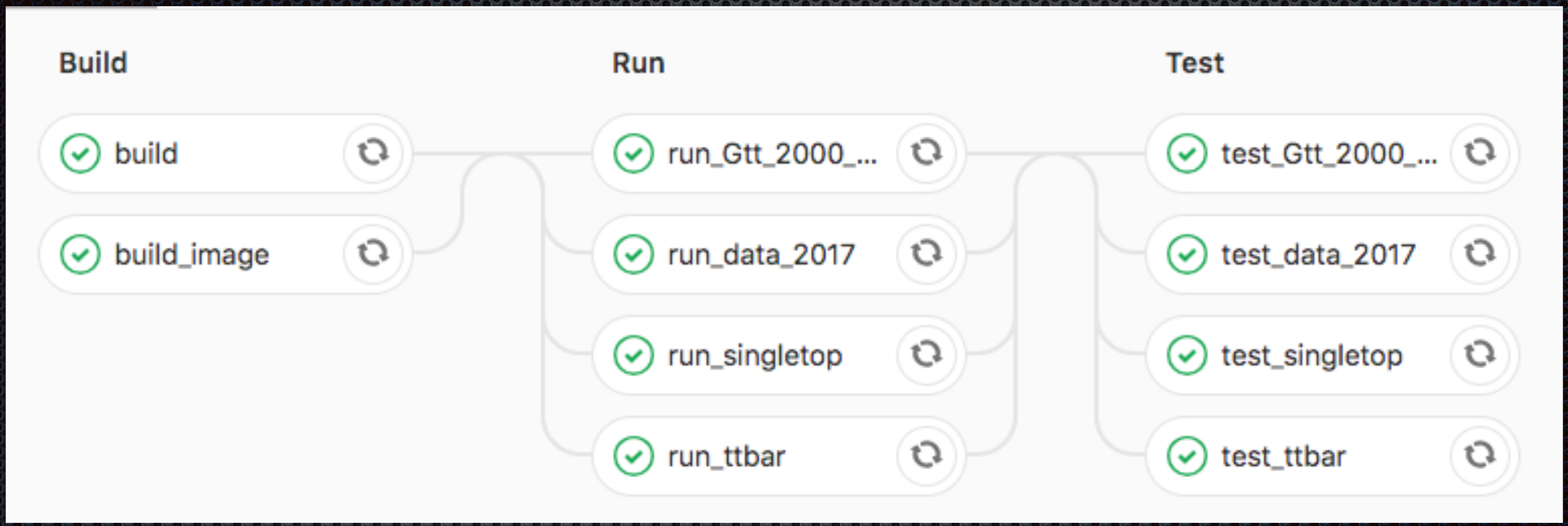
```yaml
codecapture:
  stage: build
  tags: [code-capture]
  variables:
    BUILD_BASE_IMAGE: lukasheinrich/recast_cvmfs_assisted
    TO: ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_NAME}
  image: gitlab-registry.cern.ch/runner_admin/codecapture_steering
  before_script: []
  script:
   - /codecapture_utils/steering.sh
  allow_failure: true

setup:
  stage: setup
  script:
    - rsync -a $CI_PROJECT_DIR $BUILD_DIR_ABS  --exclude $BUILD_DIR
    - grep "#!" "${PACKAGE_NAME}/README.rst" | sed -r 's/#!//g' |
    - grep "#!" "${PACKAGE_NAME}/tests/README.rst" | sed -r 's/#!!
  artifacts:
    paths:
      - $BUILD_DIR
    name: "${BUILD_DIR}_${CI_BUILD_STAGE}"
    expire_in: 6 mos
  tags:
    # Make you job be executed in a shared runner that has CVMFS mo
    - cvmfs

build:
  stage: build
  dependencies: [setup]
  script:
    - source INSTALL
  artifacts:
    paths:
      - $BUILD_DIR
    name: "${BUILD_DIR}_${CI_BUILD_STAGE}"
    expire_in: 6 mos
  tags:
```

.gitlab-ci.yml

Validate it: https://gitlab.cern.ch/ci/lint

gitlab://MultiBJets/MBJ_Analysis

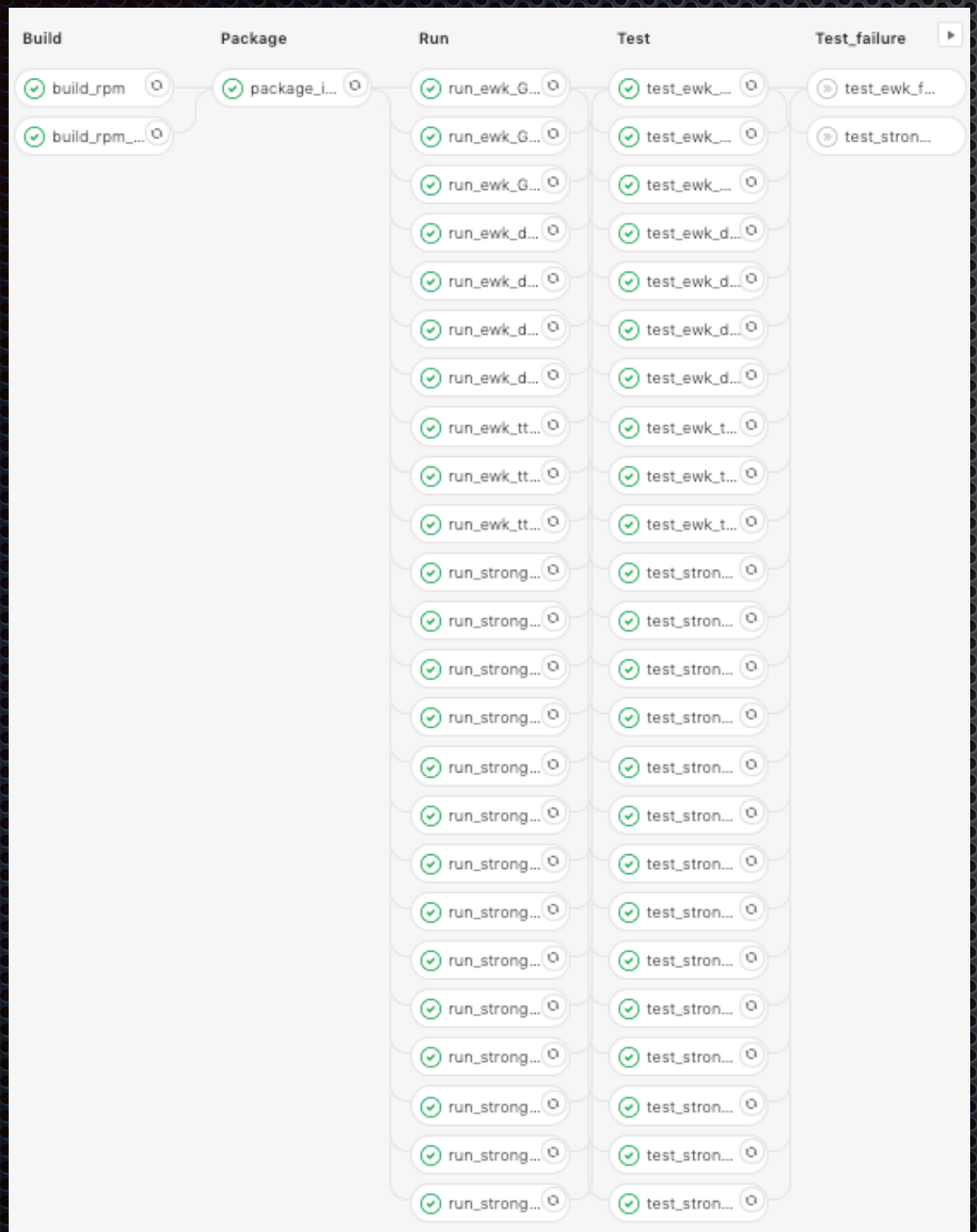
Setting up CI (II)



A workflow diagram of what your CI testing could look like

gitlab://MultiBJets/MBJ_Analysis

Setting up CI (III)



MBJ_Analysis has one of the most fleshed out analysis CIs

24 different datasets tested in under 6 minutes (240k events)

Our CI is documented here: https://indico.cern.ch/event/807536/timetable/

gitlab://MultiBJets/MBJ_Analysis/

GitLab CI Setup (I)

- Let's use MBJ's release 21 setup as an example to demonstrate how you can speed up development

```
stages:
  - build          cmake, make, and other build steps
  - scaffold       env. set up, analyze a sample root file
  - test           unit tests, cutflow checks, etc…
```

Define general stages

```
variables:
  GIT_SUBMODULE_STRATEGY: recursive
  GIT_SSL_NO_VERIFY: "true"
  PACKAGE_NAME: MultibjetsAnalysis
  SRC_DIR: src
  BUILD_DIR: build
  SRC_DIR_ABS: "${CI_PROJECT_DIR}/${SRC_DIR}"
  BUILD_DIR_ABS:  ${CI_PROJECT_DIR}/${BUILD_DIR}"
```

tell gitlab to clone submodules too

needed for CERN runners

Define global variables
see predefined variables

gitlab://MultiBJets/MBJ_Analysis/

GitLab CI Setup (II)

- Define base docker image and common setup

```
.analysis_image: &image
  image: atlas/analysisbase:21.2.27
  tags:
    - cvmfs
before_script:
  - pwd
  - ls
  - echo "Project Directory    ${CI_PROJECT_DIR}"
  - echo "Source Directory     ${SRC_DIR_ABS}"
  - echo "      Directory Name ${SRC_DIR}"
  - echo "Build Directory      ${BUILD_DIR_ABS}"
  - echo "      Directory Name ${BUILD_DIR}"
  - source /home/atlas/release_setup.sh
  - echo $SERVICE_PASS | kinit $CERN_USER
```

Define docker image

use runners with CVMFS

Setup script, environment, and kinit for data access

gitlab://MultiBJets/MBJ_Analysis/

GitLab CI Setup (III)

- Define templates for running and testing

```
.run_template: &run
  <<: *image
  stage: run
  dependencies: [build]
  script:
    - source "${BUILD_DIR}/${AnalysisBase_PLATFORM}/setup.sh"
    - MBJ_run.py --files "root://eosuser.cern.ch//eos/user/g/
gstark/MBJ/test/${SAMPLE}" --inputSource xrootd --driver
direct --submitDir submitDir --nevents 10000 ${CONFIGS}
  artifacts:
    paths:
      - submitDir
    name: "submitDir_${CI_JOB_NAME}"
    expire_in: 6 mos

.test_template: &test
  <<: *image
  stage: test
  script:
    - python tests/test.py -p ${PROCESS}
```

Define run template

can only run code if it's been built!
use artifacts from `build` *stage*

how to run the code!
variables:
- SAMPLE
- CONFIGS

save outputs for test jobs
expire in 6 months

Define test template

how to test the code!
variables:
- PROCESS

gitlab://MultiBJets/MBJ_Analysis/

GitLab CI Building

- Very easy to define compilation and build steps for your code - should match what you instruct users to do

```yaml
build_image:
  stage: build
  tags:
    - docker-image-build
  script:
    - ignore

build:
  <<: *image
  stage: build
  script:
    - mkdir $BUILD_DIR
    - cd $BUILD_DIR
    - cmake $SRC_DIR_ABS
    - make -j8
    - cd ../
    - source "${BUILD_DIR}/${AnalysisBase_PLATFORM}/setup.sh"
  artifacts:
    paths:
      - $BUILD_DIR
    name: "${BUILD_DIR}_${CI_JOB_NAME}"
    expire_in: 7d
  tags:
    - cvmfs
```

Build docker image using Dockerfile and then deploy

Build your code for run stages

Save build artifacts for run stages

gitlab://MultiBJets/MBJ_Analysis/

GitLab CI Running!

- Since you have the code built in a previous stage, grab the built code, run on a root file, and save output directory

```
run_ttbar:
  <<: *run
  variables:
    SAMPLE: mc16_13TeV.410470.PhPy8EG_A14_ttbar_hdamp258p75_nonallhad.deriv.DAOD_SUSY10.e6337_e5984_s3126_r9364_r9315_p3387
    CONFIGS: --config MultibjetsAnalysis/SUSYTools_rel21.conf --doTruth 1 --dataSource 1 --doTopPtCorrection 1 --
doTtbarHFClassification 0 --doSyst 0 --doNTUPSyst 0 --triggerLists "trigger_menu_2015,trigger_menu_2016,trigger_menu_2017"
```

use run template

```
run_singletop:
  <<: *run
  variables:
    SAMPLE: mc16_13TeV.410013.PowhegPythiaEvtGen_P2012_Wt_inclusive_top.deriv.DAOD_SUSY10.e3753_s3126_r9364_r9315_p3387
    CONFIGS: --config MultibjetsAnalysis/SUSYTools_rel21.conf --dataSource 1 --doSyst 0 --doNTUPSyst 0 --triggerLists
"trigger_menu_2015,trigger_menu_2016,trigger_menu_2017"

run_data_2017:
  <<: *run
  variables:
    SAMPLE: data17_13TeV.periodB.physics_Main.PhysCont.DAOD_SUSY10.grp17_v01_p3372
    CONFIGS: --config MultibjetsAnalysis/SUSYTools_rel21.conf --doSyst 0 --doPileup 1 --doTRF 0 --dataSource 0 --triggerLists
"trigger_menu_2015,trigger_menu_2016,trigger_menu_2017"

run_Gtt_2000_5000_200:
  <<: *run
  variables:
    SAMPLE: mc16_13TeV.375884.MGPy8EG_A14N_GG_ttn1_2000_5000_200.deriv.DAOD_SUSY10.e6351_e5984_a875_r9364_r9315_p3387
    CONFIGS: --config MultibjetsAnalysis/SUSYTools_rel21.conf --doTruth 0 --dataSource 1 --doTopPtCorrection 0 --
doTtbarHFClassification 0 --doSyst 0 --doNTUPSyst 0 --triggerLists "trigger_menu_2015,trigger_menu_2016,trigger_menu_2017"
```

gitlab://MultiBJets/MBJ_Analysis/

GitLab CI Testing! (I)

- We ran all the code, and assuming it ran fine, we want to test it for physics! (and other boring stuff)

```
test_ttbar:
  <<: *test
  dependencies: [run_ttbar]
  variables:
    PROCESS: ttbar

test_singletop:
  <<: *test
  dependencies: [run_singletop]
  variables:
    PROCESS: singletop

test_data_2017:
  <<: *test
  dependencies: [run_data_2017]
  variables:
    PROCESS: data_2017

test_Gtt_2000_5000_200:
  <<: *test
  dependencies: [run_Gtt_2000_5000_200]
  variables:
    PROCESS: Gtt_2000_5000_200
```

use test template

depends on

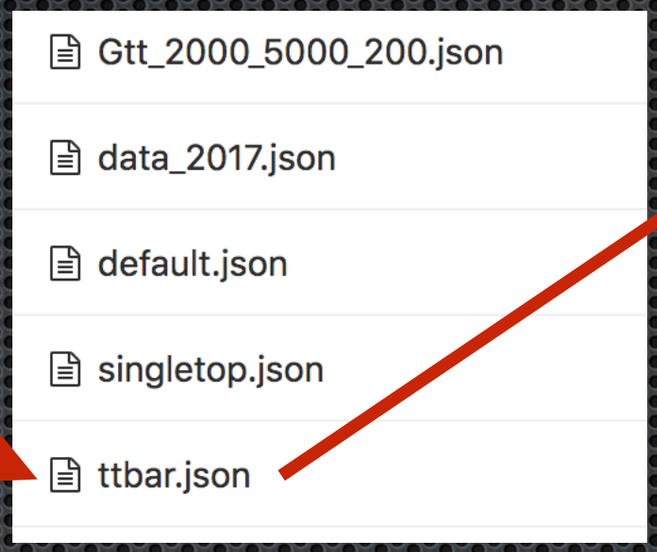
```
run_ttbar:
  <<: *run
  variables:
    SAMPLE: mc16_13TeV.
410470.PhPy8EG_A14_ttbar_hdamp258p75_nonallhad.deriv.DAOD_SUSY10
.e6337_e5984_s3126_r9364_r9315_p3387
    CONFIGS: --config MultibjetsAnalysis/SUSYTools_rel21.conf --
doTruth 1 --dataSource 1 --doTopPtCorrection 1 --
doTtbarHFClassification 0 --doSyst 0 --doNTUPSyst 0 --
triggerLists
"trigger_menu_2015,trigger_menu_2016,trigger_menu_2017"
```

- Each process defines what values to compare against in the test (see next slide)

gitlab://MultiBJets/MBJ_Analysis/tests/test.py

GitLab CI Testing! (II)

```
test_ttbar:
  <<: *test
  dependencies: [run_ttbar]
  variables:
    PROCESS: ttbar
```



ttbar.json 194 Bytes

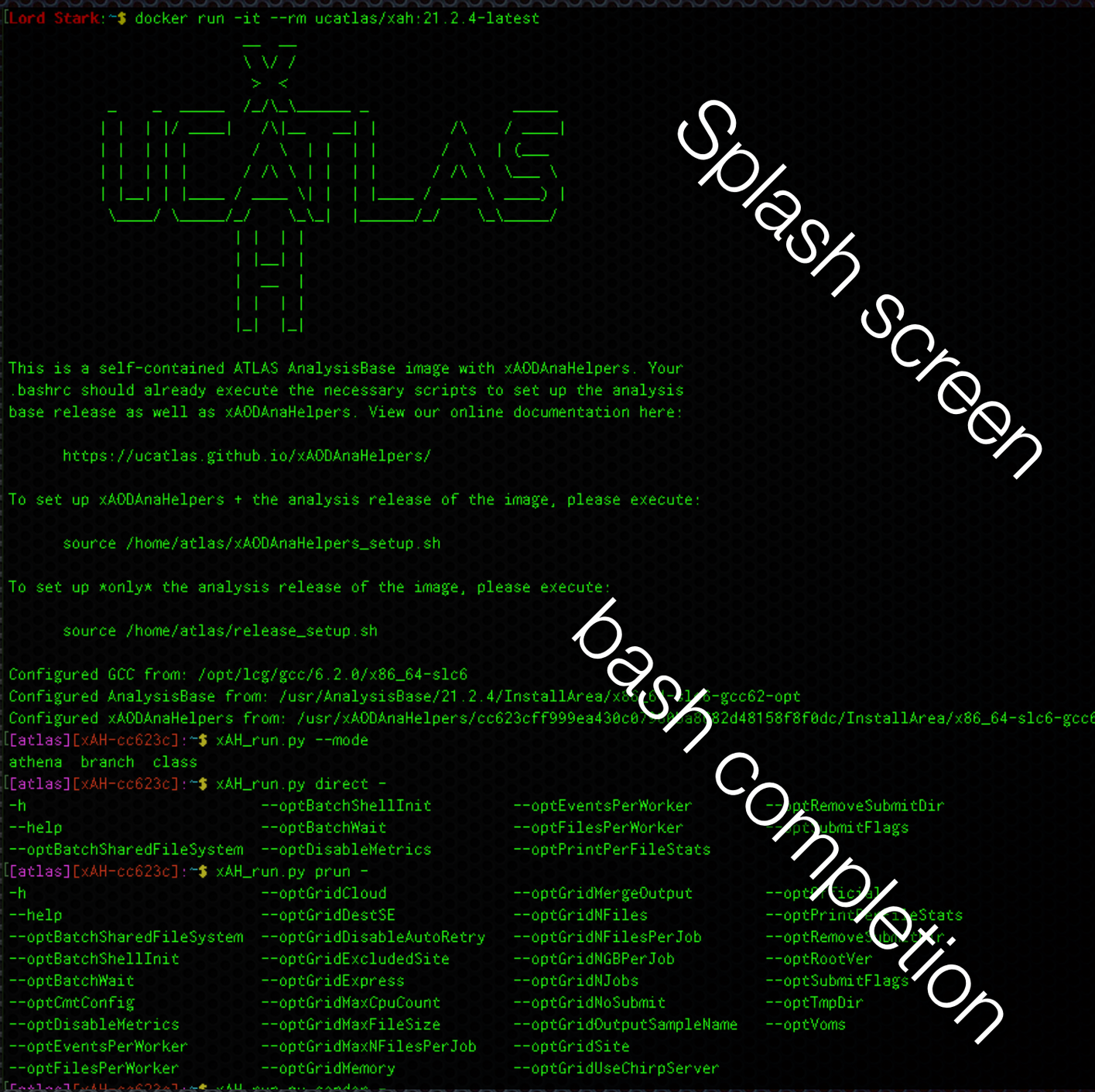
```
[
    60000.0,
    43837728.85498047,
    87616347038.0586,
    10050.0,
    0.0,
    0.0,
    0.0,
    0.0,
    8615.0,
    9978.0,
    10000.0,
    9987.0,
    3329.0,
    3329.0,
    279.0
]
```

```
def test_cutflow(self):
  '''Is the raw cutflow what we expect?'''
  names = ['metadata unweighted',
           ...
          'histfitter selection']
  expected = self.get_json('cutflow')
  cutflow = [self.f.cut_flow.GetBinContent(i) for i in range(1, 16)]

  self.assertListEqual(cutflow, expected, msg=error_msg(names, cutflow, expected))
```

- Each process defines what values to compare against in the test

Dockerizing (I)

- xAH builds docker images: https://hub.docker.com/r/ucatlas/xah/ that come with the analysis release + xAH compiled by default
  - Images are built using continuous integration
- You can run a full analysis locally (without a tier3)!
  - Working with ATLAS Connect to allow for condor submissions directly from this docker image



Try it yourself!

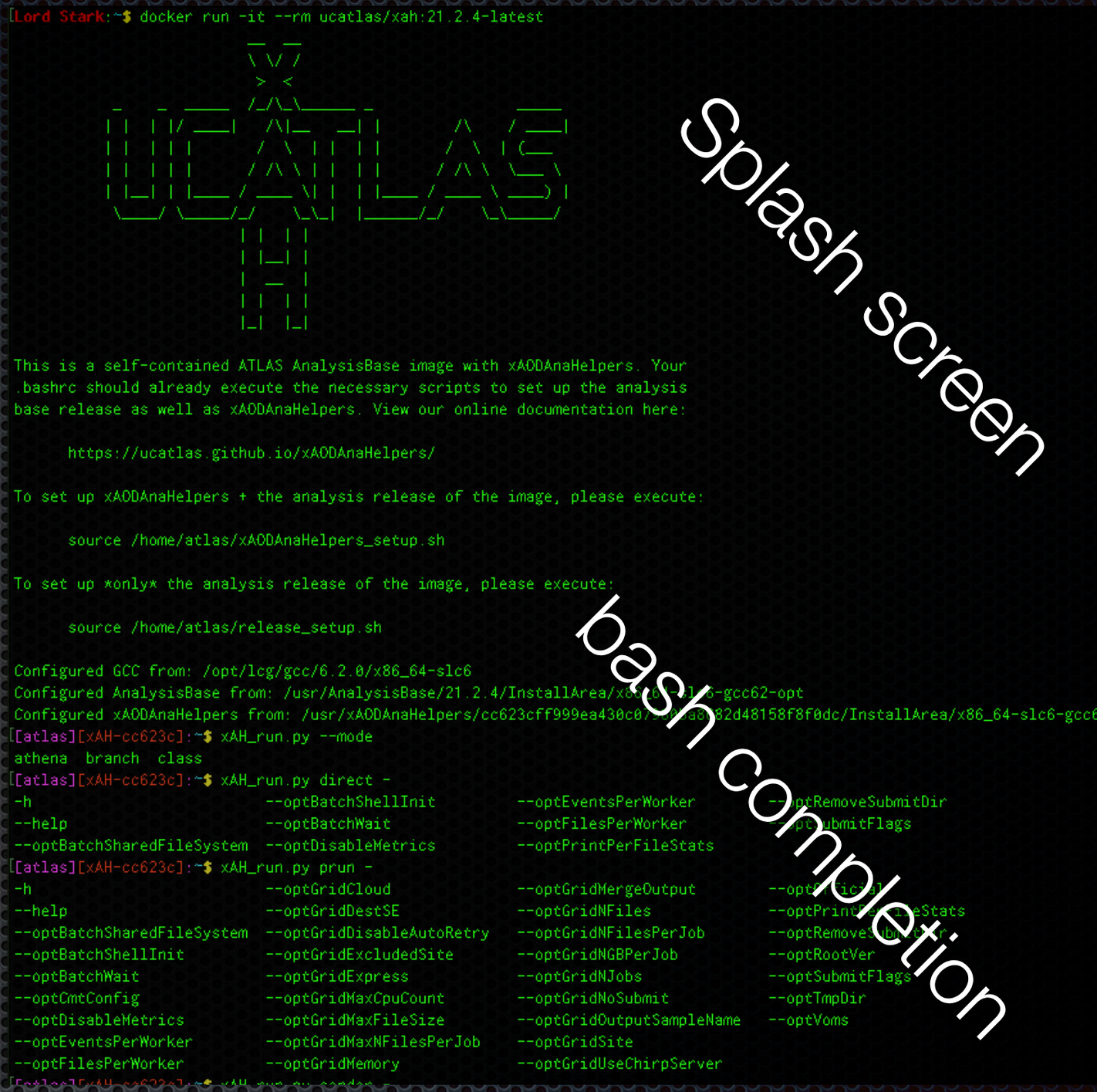
```
docker run -it --rm ucatlas/xah:21.2.18-latest
```

More details in Lukas' talk

Collaborative effort with Lukas Heinrich, NYU (docker expert in ATLAS)

Dockerizing (II)

- **Hands-on demonstration using xAODAnaHelpers**: https://gist.github.com/kratsg/0e3f6382c3f9a59c652fe87e2b1cd7eb
- Go from nothing to plots in under 5 minutes with minimal set-up!



Try it yourself!

```
docker run -it --rm ucatlas/xah:21.2.22-latest
```

More details in Lukas' talk

Collaborative effort with Lukas Heinrich, NYU (docker expert in ATLAS)

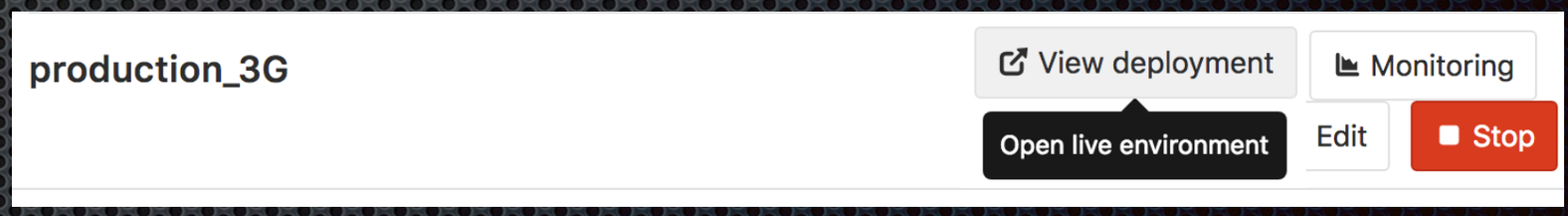
Repo: https://gitlab.cern.ch/atlas-phys-susy-wg/AnalysisSUSYToolsConfigurations

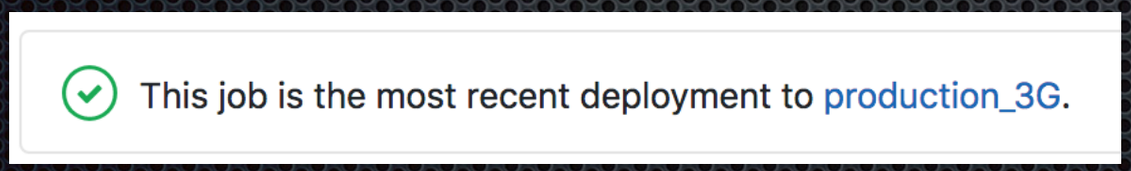
Autogenerate Twiki pages

```
.build_twiki_text:
  stage: build
  script:
    - python twikitable/generateTwikiTable.py --
analyses readme/metadata/analyses.json --filter-sub-
group ${SUBGROUP} | tee ${SUBGROUP}Combinations.twiki
  artifacts:
    paths:
      - ${SUBGROUP}Combinations.twiki
```

```
build_3G:
  extends: .build_twiki_text
  variables:
    SUBGROUP: 3G

deploy_3G:
  extends: .deploy_twiki_text
  dependencies: [build_3G]
  variables:
    SUBGROUP: 3G
```

```
.deploy_twiki_text:
  image: atlas/analysisbase
  stage: deploy
  before_script:
    - source /home/atlas/release_setup.sh
    - echo $SUSYPRODPASS | kinit $SUSYPRODACCOUNT
  script:
    - xrdcp --force ${SUBGROUP}Combinations.twiki
root://eosatlas.cern.ch//eos/atlas/
atlascerngroupdisk/phys-susy/MCProd_www/${SUBGROUP}
Combinations.twiki
  environment:
    name: production_$SUBGROUP
    url: https://cern.ch/atlas-phys-susy-mcprod/$
{SUBGROUP}Combinations.twiki
  only:
    refs:
      - master
    variables:
      - $CI_PROJECT_NAMESPACE == "atlas-phys-susy-wg"
```





Deployed to: https://atlas-phys-susy-mcprod.web.cern.ch/atlas-phys-susy-mcprod/3GCombinations.twiki
Embedded in: https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/ThirdGenerationCombinations

Summary

- ASG will officially support people putting their analysis code into **gitlab.cern.ch**
  - Expert analyzers and developers are welcome to keep code in GitHub if they already started using it before GitLab was introduced
- **Continuous Integration provides a lot of benefits such as**
  - checking that the **code can actually compile**
  - verifying that the code runs fine on a sample root file *(perhaps from EOS, but details will be hashed out in ASG meetings)*
  - test the output of running the code and **alerting you to changes in cutflow or physics**
  - **compiling PDFs** for supporting notes, CONF notes, papers, or TDRs/FDRs/etc…
  - **generating static documentation** for your analysis
  - static code analysis techniques
  - allowing for **custom deployments** such as:
    - creating a docker image of the compiled analysis code and uploaded to either hub.docker.com or GitLab's docker registry (more details in Lukas' talk)
    - uploading documentation to a website or a new page
    - trigger analysis reinterpretation through RECAST
    - **[expert]** automatically submitting jobs to Panda or Condor or batch nodes
    - **[expert]** automatically updating CDS with the newly created PDF when adding a new tag

Backup

Automation example in SUSY Combinations

Org-auto-nization

A key goal is to automate as much of this process as possible

- Three key repositories right now:
  - atlas-phys-susy-wg/AnalysisSUSYToolsConfigurations
    - Upload SUSYTools (ST) config files here
  - atlas-phys-susy-wg/Combinations/readme
    - Keep track of organizational/metadata information (analyses, contacts, glance IDs, subgroup, etc…)
  - atlas-phys-susy-wg/Combinations/twikitable
    - First of many scripts — this one generates a twiki page comparing specified ST config files
- Expect contacts to upload their SUSYTools config files
  - Triggers a pipeline which clones Combinations/readme and Combinations/twikitable to build twiki pages showing overlaps/differences in ST configurations
- Expect subgroup conveners to inform us of issues with organizational metadata
  - Changes to metadata trigger a pipeline which updates the wiki (https://gitlab.cern.ch/atlas-phys-susy-wg/Combinations/readme/wikis/home)
- Expect (bug?)fixes and improvements to various scripts such as twistable
  - Updates trigger a pipeline which checks that the script runs against the existing metadata information in “readme” and ST config files in “ASTC”

https://gitlab.cern.ch/atlas-phys-susy-wg/Combinations/readme/wikis/analysis-contacts

Analysis Contacts page

analyses.json 10.2 KB

```
[
    {
        "analysis": "all hadronic",
        "config_file": "{glance}.conf",
        "contact": "David Miller",
        "glance": "ANA-SUSY-2018-11",
        "is3G": false,
        "isEWK": true,
        "isRPVLL": false,
        "isStrong": false,
        "target": "C1N2 --> Wh/WZ (2b2j and 4j), C1C1 --> WW (4j), GGM Zh/ZZ (2b2j and 4j)"
    },
    {
        "analysis": "1L",
        "config_file": "{glance}.conf",
        "contact": "Eric Schanet",
        "glance": "ANA-SUSY-2018-10",
        "is3G": false,
        "isEWK": true,
        "isRPVLL": false,
        "isStrong": false,
        "target": "C1N2 --> WZ/Wh, C1C1 --> WW with 1 lepton and (b-)jets (+ strong production)"
    },
    {
        "analysis": "staus",
        "config_file": "{glance}.conf",
        "contact": "Chenzheng Zhu",
        "glance": "ANA-SUSY-2018-04",
        "is3G": false,
        "isEWK": true,
        "isRPVLL": false,
        "isStrong": false,
        "target": "direct staus, final states
    },
    {
        "analysis": "2L0J",
        "config_file": "{glance}.conf",
        "contact": "Holly Pacey",
        "glance": "ANA-SUSY-2018-32",
        "is3G": false,
        "isEWK": true,
        "isRPVLL": false,
```

Continuous Integration

atlas-phys-susy-wg › Combinations › readme › Wiki › **Analysis contacts**

Analysis contacts

Last edited by **SUSY Combinations** 8 hours ago

Analysis Contacts

To make an update to this page, edit Combinations/readme/metadata/analyses.json.

This page is automatically generated by a CI job in Combinations/readme. Do not edit this page manually.

Analysis	Contact	Glance	Config	3G	EWK	RPVLL	Strong
all hadronic	David Miller	ANA-SUSY-2018-11	ANA-SUSY-2018-11.conf		X		
1L	Eric Schanet	ANA-SUSY-2018-10	ANA-SUSY-2018-10.conf		X		
staus	Chenzheng Zhu	ANA-SUSY-2018-04	ANA-SUSY-2018-04.conf		X		
2L0J	Holly Pacey	ANA-SUSY-2018-32	ANA-SUSY-2018-32.conf		X		
2L2J		ANA-SUSY-2018-05	ANA-SUSY-2018-05.conf		X		
compressed	Jeff Shahinian	ANA-SUSY-2018-16	ANA-SUSY-2018-16.conf		X		
	Tina Potter	ANA-SUSY-2018-02	ANA-SUSY-2018-02.conf		X	X	
3L off-shell	Lucia Pedraza	ANA-SUSY-2018-06	ANA-SUSY-2018-06_offshell.conf		X	X	
	Lucia Pedraza	ANA-SUSY-2018-06	ANA-SUSY-2018-06_onshell.conf		X	X	
					X	X	
					X		
					X		X
			ANA-SUSY-2018-09.conf	X	X	X	X
							X

.gitlab-ci.yml 527 Bytes

.gitlab-ci.yml

```
stages:
  - build
  - run
  - test
  - docs

update_wiki:
  stage: docs
  script:
    - git clone https://${GITLAB_USER}:${GITLAB_ACCESS_TOKEN}@gitlab.cern.ch/atlas-phys-susy-wg/Combinations/readme.wiki.git
    - python ci/update_wiki.py
    - cd readme.wiki
    - git config user.name "SUSY Combinations"
    - git config user.email atlas-phys-susy-combinations-conveners@cern.ch
    - |
        if ! git diff-index --quiet HEAD; then
          git add *.md
          git commit -m "update wiki"
          git push
        fi
```

https://twiki.cern.ch/twiki/bin/view/AtlasProtected/ThirdGenerationCombinations

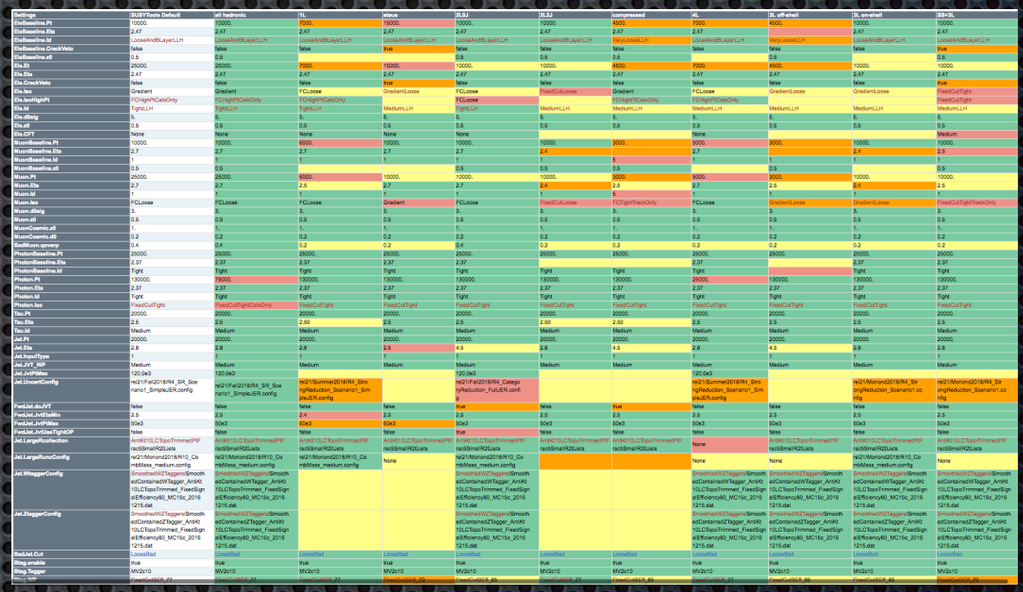
https://twiki.cern.ch/twiki/bin/view/AtlasProtected/ElectroweakCombinations

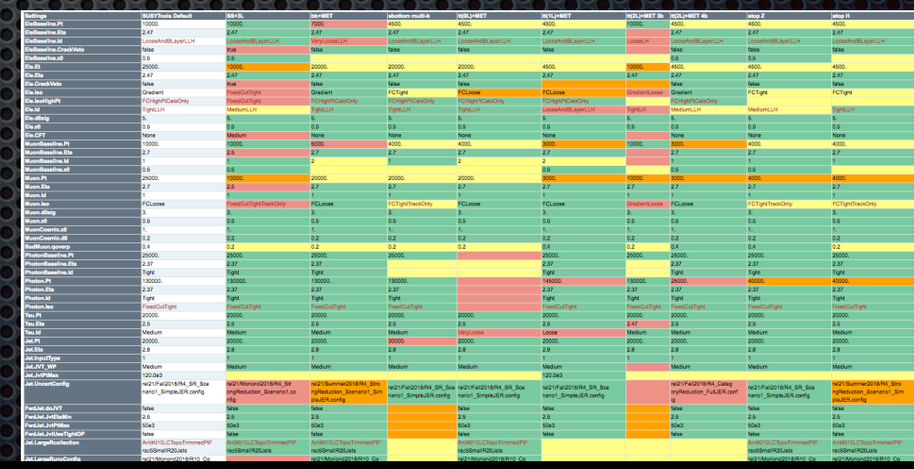
ST Config Comparisons

analyses.json 10.2 KB

```
[
    {
        "analysis": "all hadronic",
        "config_file": "{glance}.conf",
        "contact": "David Miller",
        "glance": "ANA-SUSY-2018-11",
        "is3G": false,
        "isEWK": true,
        "isRPVLL": false,
        "isStrong": false,
        "target": "C1N2 --> Wh/WZ (2b2j and 4j), C1C1 --> WW (4j), GGM Zh/ZZ (2b2j and 4j)"
    },
    {
        "analysis": "1L",
        "config_file": "{glance}.conf",
        "contact": "Eric Schanet",
        "glance": "ANA-SUSY-2018-10",
        "is3G": false,
        "isEWK": true,
```

```
build_3G:
  extends: .build_twiki_text
  variables:
    SUBGROUP: 3G

deploy_3G:
  extends: .deploy_twiki_text
  dependencies: [build_3G]
  variables:
    SUBGROUP: 3G
```





EWK

3G

Continuous Integration

```
python twikitable/generateTwikiTable.py --analyses readme/metadata/analyses.json
--filter-sub-group ${SUBGROUP} | tee ${SUBGROUP}Combinations.twiki
```