Lesson 1 : Course Prerequisites

Bruce Becker, CSIR Meraka Institute
bbecker@csir.co.za

Sci-GaIA Winter School

www.sci-gaia.eu
Preface to the school

- Some considerations before you start
- Continuous Integration
- Tools you will need
- Getting started with git and Github
- Getting started with Docker and Docker Hub
Some considerations before you start

- This is just the beginning
- You will not be working alone
- Things will go horribly wrong
- If something is worth doing, it's worth keeping
- If you haven't expressed it, it doesn't exist
- If you can't reproduce it, it doesn't exist
- If you haven't tested it, it doesn't work
- You are a human – do as little as possible!
This is just the beginning: Code for the future

Congratulations! You have been accepted to join the Sci-GaIA winter school on application development for science gateways. During this school, we will teach you skills and methods which are current best practice, and technology which is still evolving.

Some things to keep in mind:

- Whatever you do today will probably need to change in 1-2 years
- You will likely have a burst of activity during this school, and shortly afterwards, while you develop your application. Some time later, you may come back to the code that you've written. How will you know what it is or does, or whether it even works?
- Be conscious of the fact that future you will be very different from you
You will not be working alone: Code for humans everywhere

You have submitted a proposal as a team and will be developing code as a team, during the school. Whatever code or documentation you write needs to be as clear and explicit as possible, so that your team mates are able to understand it.

The school staff will also need to understand your work, without the luxury of being side-by-side and the ability for non-verbal communication. Be aware that what you do may be misinterpreted or misunderstood and try to write as many comments explaining why you are doing things instead of what you are doing.

You are also not able to predict who will be interested in this project or when.

- Put your code into a repository so that collaborators and teammates can access it any time.
- Code review is a good means of identifying not only bugs, but vague or unclear coding.
Things will go horribly wrong

Being optimistic does not pay off in this game.

The more you assume about the environment in which your code will execute, the more bugs you will introduce into the application.

You cannot be trusted
If you haven't expressed it, it doesn't exist

There is a great temptation to make quick changes and hack away at what may seem like a small problem until it finally goes away. However, the important aspect of solving a problem is not that it is solved, but how you solved it. This implies a series of tasks or actions, which it is better to have expressed explicitly – you will probably need to reproduce this solution at some point.

Remember that whatever you do on the console or interactively is ephemeral and therefore is adversarial to both your future self as well as your current collaborators.

Use an automation tool wherever possible
If you can't reproduce it, it doesn't exist

The final product (application) is less valuable than the action or steps which were taken to achieve it.

As part of the lesson exercises and evaluation, we will be checking the functionality of the executables your team produces, however we will also be running automated tests on the code you've written for producing it.

“Magic” executables, which spring into being inexplicably, are not a good way to collaborate and demonstrate knowledge of the framework which they were designed for.

• Ensure that you provide functionality for reproducing certain expected states of the code, to convince yourself, your teammates and the school staff that you know what you are doing.
If you haven't tested it, it doesn't work

Would you buy something from a person who said “just trust me”, when asked if their product works? Probably not. You should never take functionality of an application at face value and explicitly test whether it works as advertised in an environment which looks as much as possible like the final deployment.

- Automated testing will be done by the school staff on every commit of your code, and the results will be transparently published to both you and the staff.

- Pre-defined compilation and sanity tests will be provided by the school staff which your application has to pass to avoid breaking the deployment infrastructure.

- The more functional and unit tests you implement, the more confidence you'll have in your application and the less time you'll spend on debugging.
You are human – do as little as possible

Many of the tasks we've alluded to – code integration, review, testing, deployment, etc – are **repetitive** and quite simple to **automate**. We will be using automation wherever possible to perform these tasks, leaving you, the developer free to work on the most important aspect of your project: **the code**.

This requires some setup initially, which may seem like a bit of unnecessary overhead. Bear in mind that it is a once-off investment of time and will help you to work in a more open, collaborative way, since there are always temptations to follow “**anti patterns**”

There is also a price to be paid for not automating, which is part of the **Technical Debt** of your project.
Continuous Integration

• When working in a team, you want to work as smoothly and as independently as possible

• Continuous Integration is a paradigm of software development which makes use of automated compilation and testing to know the state of the application at any point.

• Allows you to:
  - Be aware of the end result of any change introduced to the code
  - Ensure that there is an explicit expression of any errors
  - Explicitly check that tests are being run, what their state is and whether the application can be deployed
Best practices

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
- Keep the build fast
- Test in a clone of the production environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

https://www.thoughtworks.com/continuous-integration

https://en.wikipedia.org/wiki/Continuous_integration#Best_practices
How do you implement these best practices?

Maintaining the source code repository and implementing some of the self-tests are one thing, but

- how do you automate the build and testing?
- How do you run this in a clone of the production environment?

Turns out there's a whole team working behind the scenes to support your work!
Meet the team

Courses usually have **students** and **lecturers**. However as with most courses, there are a number of assistants needed to ensure the course runs smoothly. In our case, they are built in software...

Student

Lecturer

Robots
Automation, Verification and Transparency

Continuous Integration services are deployed for the specific purposes of the course. These are built with automation and orchestration tools.

**Ansible**: service configuration, deployment and orchestration

**Jenkins**: automated testing and integration of student code

This provides contextual links between the *code* you write and the *state* of the executable, helping you and the tutors to identify where bugs or errors have been introduced.
Automation, Verification and Transparency
Tools you will need

- Development machine – suggest 🍒 (🍎 🍒 - YMMV)
- Knowledge of the shell
- A good editor/IDE
- Github
- Docker Hub
Not covered in this course

If you need to brush up on your skills or knowledge before the course, we suggest following the **Software Carpentry Lessons:**

- Using the shell, and writing shell scripts
Github
Github: Needs no introduction
https://github.com/

- Any version control system would be sufficient in principle, but we choose GitHub:
- Not just a hosted git platform:
  - Great user interface
  - Social network of developers
  - Extra tools and excellent API: developer.github.com/v3
  - Good relationship with academic activities – education.github.com
Practical: Set up Git on your development machine

- apt-get install git
- yum install git
- Others: [git-scm.com/downloads](http://git-scm.com/downloads)
Software Carpentry Git course
swcarpentry.github.io/git-novice/02-setup

- Install git on your development environment (laptop/desktop)
- Configure git for first time use
- Full video:
  - 🎥 https://youtu.be/hKFNpxxkbO0
Git is a very powerful tool which allows almost any kind of version control workflow. This can make it confusing for new users.

Github flow is a “good practice” guide to collaborating and contributing to software development using Git.

Github flow inherently includes many concepts from continuous integration (code review, automated testing, etc).
Practical

- Read the Github Flow help article:
  - guides.github.com/introduction/flow
- Familiarise yourself with the terminology
- Watch the Github Training videos
- 💻 youtube.com/GithubGuides
Practical: Create Your Github Account

- Dashboard
- Profile
- Account Settings
Dashboard: github.com
Profile: github.com/<username>
Account Settings: github.com/settings/profile
Practical: Create your first repository - planets

- Create a local repository on your development machine
  - swcarpentry.github.io/git-novice/03-create.html
- Add your changes
  - swcarpentry.github.io/git-novice/04-changes
- Create a remote repo on Github and merge with your local repo
  - swcarpentry.github.io/git-novice/07-github
Docker
Linux Containers and Docker

- Linux containers are excellent tools for application development and testing
- Docker provides a full toolset to develop, test, ship and share containers
- We will be using Docker to:
  - Prepare development environment
  - Test in pre-defined environments
  - Validate student work
Containers are not Virtual Machines
Practical : Install Docker

docs.docker.com/engine/installation/
Docker Hub

- Docker Hub is a place to publish and share application containers
- With Docker Hub, containers can be:
  - Easily referenced
  - Version controlled
  - Forked, re-worked
- Ensure that your containers are always available:
  - Build and check containers locally - `docker build ; docker run`
  - Push them to docker hub - `docker push ; docker commit`
- Docker Hub can automatically build containers by linking a Github repo to a Docker repo
Practical: Create your account on docker hub
hub.docker.com
Practical: Create the repository for a docker container

- Log into docker hub
  - `docker login`
- Run hello world
  - `docker run hello-world`
- Pull an ubuntu image
  - `docker pull ubuntu`
Practical : Work with containers

docs.docker.com/engine/quickstart
Docker Engine – docker files

# Ubuntu 15.10 Ansible-ready docker file
FROM ubuntu:15.10
MAINTAINER Bruce Becker bbecker@csir.co.za
# Get git
RUN apt-get update
RUN apt-get install -y git build-essential
#
RUN git clone git://github.com/<username>/planets
WORKDIR planets
RUN cat mars.txt
Practical: Automate a build

- Create repo in github
- Add Dockerfile
- Create new automated build
Docker hub dashboard: [hub.docker.com](https://hub.docker.com)

### Repositories

<table>
<thead>
<tr>
<th>Repository</th>
<th>Stars</th>
<th>Pulls</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>brucellino/invenio</code></td>
<td>0</td>
<td>56</td>
</tr>
<tr>
<td><code>brucellino/grid-ui-ansible-ready</code></td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td><code>brucellino/zenodo</code></td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td><code>brucellino/zenodo-docker-role</code></td>
<td>0</td>
<td>18</td>
</tr>
</tbody>
</table>
Create Autobuild from Github

Create Auto-build

Github

Link Account

Bitbucket
Further reading

- Docker Engine Guide:
  - docs.docker.com/engine/userguide
- Dockerfile Reference:
  - docs.docker.com/engine/reference/builder
- Docker Self-Paced Training:
  - training.docker.com/self-paced-training