
CyVerse Documentation

Release 0.2.0

CyVerse

Jan 05, 2023

Contents

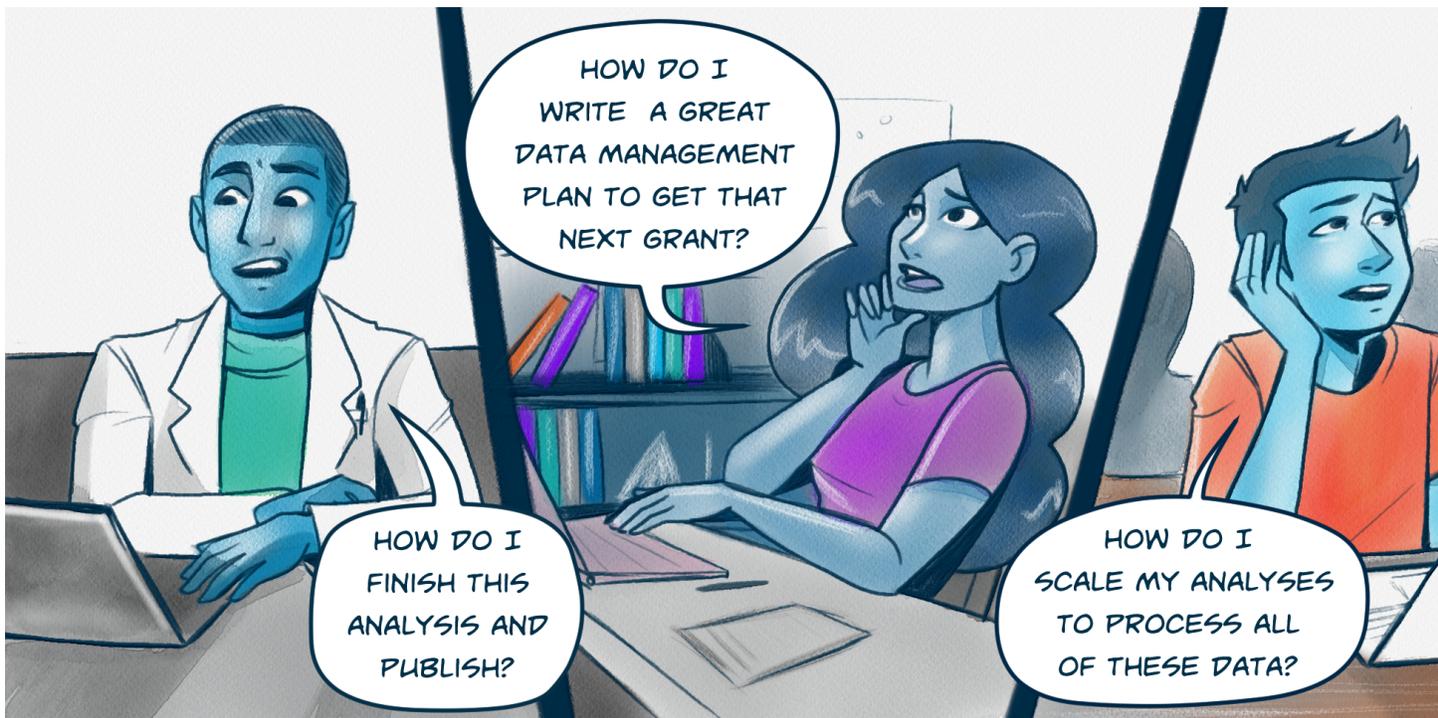
1	Expected outcomes:	3
2	Funding and Citations	167
3	License	169



 Learning Center Home

Foundational Open Science Skills (FOSS) is a novel, camp-style training designed to prepare principal investigators and their lab teams, both new and established, to meet the growing expectations of funding agencies, publishers, and research institutions for scientific reproducibility and data accessibility.

There are no pre-requisites for FOSS, but the course will cover a lot of material in a short time. Participants who have limited computational experience should try to view the [Software Carpentry Core Lessons](#) before attending.



Expected outcomes:

- Become familiar with productivity software for organizing your lab group, communications, and research
- Learn how to scale out your computation from laptop to cloud and high performance computing (HPC) systems
- Learn how to manage data for open science and reproducibility

By working through an example project relevant to their interests, participants will practice open science skills using CyVerse, GitHub, R or Python, and other resources. At the end of the week, students will present a plan for how to integrate open science into their labs.



 Learning Center Home

1.1 Before FOSS Starts

Please endeavor to complete the [pre-FOSS setup](#) before arriving at FOSS.

FOSS runs under a [Code of Conduct](#). Please familiarize yourself with it.

FOSS Online sessions are recorded and recordings are posted on CyVerse's private YouTube channel for participants to review at any time (only those with the link can view).

Need help?

Couldn't find what you were looking for?

- You can talk to any of the instructors or TAs if you need immediate help.

- Chat with us on [CyVerseFOSS Slack](#).
- Post an issue on the documentation [issue tracker](#) on GitHub



Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



1.2 Pre-FOSS Setup

Please set up the Prerequisite accounts and software to prepare for FOSS Online; help setting up both Prerequisites and Optional Downloads will be available during Onboarding.

Prerequisite	Notes	Links
Wi-Fi-enabled laptop	You should be able to use any laptop (Windows/MacOS/Linux.). We strongly recommend Firefox or Chrome as your browser. It is recommended that you have administrative/install permissions on your laptop.	<ul style="list-style-type: none"> • Download FireFox • Download Chrome
Zoom Account	Please ensure that you have a CyVerse account and have verified your account by completing the verification steps in the email you got when you registered.	Register for your Zoom account at http://www.zoom.us/ .
CyVerse Account	Please ensure that you have a CyVerse account and have verified your account by completing the verification steps in the email you got when you registered.	Register for your CyVerse account at http://user.cyverse.org/ .
GitHub Account	Please ensure that you have a GitHub account if you don't have one already	Register for your GitHub account at https://github.com/ .
Dockerhub Account	Please ensure that you have a Dockerhub account if you don't have one already	Register for your Dockerhub account at https://hub.docker.com/ .
Text Editor	Please ensure that you have a Text Editor of your choice. Any decent Text Editor would be sufficient. Recommended ones include SublimeText and Atom (more in Optional Downloads below)	Register for Sublime at https://www.sublimetext.com/ . Register for Atom at https://atom.io/ .
Slack for networking	We will be using Slack extensively for communication and networking purposes, especially between online class sessions	Register for Slack at https://slack.com/ .
ORCID ID	If you haven't set up an Open Researcher and Contributor ID (ORCID) yet, please consider doing so (and add it to your CyVerse account profile)	Register for ORCID at https://orcid.org/ .

Optional

Below are some other downloads that are not required for FOSS, but which provide some options for functionalities we will cover.

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



1.3 Schedule

Important: Please fill out the weekly lesson feedback [Google Form](#)

Note: Times are shown in Arizona Time GMT-7 (same as Pacific Daylight Time until November)

Date (Week)	Time	Activity (Instructor)	Content	Expected Outcomes
July 28th 30th (Week 0 - Onboarding)	Tuesday 11:00 am Thursday 1:00 pm	Welcome (Swetnam)	Overview of Code of Conduct, Course Materials, and logistics	
	Tuesday 11:30 AM Thursday 1:30 PM	Discussion: What is Open Science? (Merchant)	Overview of CyVerse mission	Understand why we teach FOSS
	Tuesday 12:00 PM Thursday 2:00 PM	Setup profile and accounts (Swetnam)	QUBES Hub Faculty Mentoring Network , CyVerse , Slack	Get familiar with our virtual platforms for learning and creating material
	Tuesday 12:30 PM Thursday 2:30 PM	CyVerse Walkthrough (Cooksey)	Navigating the Portal, Discovery Environment, Data Store, and other platforms	Optional: Launching executable and interactive apps. Add a tool and app
		Homework for Week 1	Introduce yourself via QUBES Profile	Learn about how to navigate and operate on the QUBES website

Date (Week)	Time	Activity (Instructor)	Content	Expected Outcomes
1	August 11 11:00 am August 13 1:00 pm	Key Concepts: Communication & Documentation (Swetnam slides)	Using tools like Slack, Medium, Twitter	Project management tools, and public relations with social media.
	11:45 am 1:45 pm onward	Breakout Sessions	Communication & Project Management Tools	Discuss how you can use GitHub, Websites, and Project Management Tools to empower your open-science lab

Date (Week)	Time	Activity (Instructor)	Content	Expected Outcomes
2	August 25 11:00 am August 27 1:00 pm	Essential Skills	Starting with key concepts of Linux command line and Git	
		Key Concepts: Linux (Swetnam)	GitPitch link Why Linux, Version Control, Containers	Understanding what open source software means to open science
	13:00-14:00	The Research Object I: Introduction to GitHub (Balk)		a basic understanding of <code>git</code>
			Optional: Command Line Refresher	Basic navigation and file manipulation
	09:20-10:00	FAIR Data Principles (Walls)	Metadata, standards, licensing, legal, and ethics	Be able to find standards for your data, choose a license

Date (Week)	Time	Activity (Instructor)	Content	Expected Outcomes
3	September 8 11:00 am September 10 1:00 pm	Planning Your Open Science Lab	Making collaboration and project management efficient	

Date (Week)	Time	Activity (Instructor)	Content	Expected Outcomes
4	September 22 11:00 am September 24 1:00 pm	Data Management Overview	Organization, Sharing, Metadata	Understand why you need data management
		Data Management Plans (Walls)	Writing a Data Management Plan	Learn how to write an excellent DMP!
		Data Management Tools (Walls)	Open Science Framework, Protocols.io, CyVerse DataCommons	Be able to work with data management tools

Date (Week)	Time	Activity (Instructor)	Content	Expected Outcomes
5	October 6 11:00 am October 8 1:00 pm	Reproducible Research: Basics	Turning tools into a workflow	
		Introduction to Cloud and HPC (Swetnam)	OpenScienceGrid, XSEDE, HPC, Commercial Services	
		Reproducible Research I : Launching VMs (Williams)	Launch a virtual machine in Atmosphere and provision it	
		Reproducible Research II: Provisioning, GitHub and Licensing (Williams)		

Date (Week)	Time	Activity (Instructor)	Content	Expected Outcomes
6	October 20 11:00 am October 22 1:00 pm	Reproducible Research: Containers	Developing and using your tools on any platform	Basics of Docker, creating your own containers
		The Research Object I: Introduction to Containers (Cooksey)	Intro to BioContainers, Docker, Singularity, and Kubernetes	Search for, locate, and launch a container anywhere
		Reproducible Research III: Containers and workflows		
		Reproducible Research V: Build your own Containers (Williams, Swetnam)	Basics of Docker, creating your own containers	

Date (Week)	Time	Activity (Instructor)	Content	Expected Outcomes
7	November 3 11:00 am November 5 1:00 pm	Reproducible Research: Continuous Integration	Making things reliable and automated	
		Reproducible Research IV: Computational Notebooks in CyVerse (Tuteja)	Connecting your work in CyVerse VICE	Learn to create a VICE app

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



1.4 Instructors

Amanda Cooksey, Science Informatician, CyVerse | amcooksey@cyverse.org

Eric Lyons, CyVerse Co-PI and UA Science Lead | ericlyons@email.arizona.edu

Nirav Merchant, Data Science Director | nirav@email.arizona.edu

Tyson Swetnam, Research Assistant Professor, University of Arizona | tswetnam@cyverse.org

Reetu Tujeta, Science Informatician, CyVerse | reetututeja@cyverse.org

Jason Williams, Training Lead, Cold Spring Harbor Laboratory | williams@cshl.edu

1.4.1 CyVerse Staff

Shelley Littin Coordinator, Marketing & Communications | littin@cyverse.org

Tina Lee, User Engagement Officer | tinal@cyverse.org

Mary Margaret Sprinkle, Assistant Director Finance & Management | marys@cyverse.org

Mariah Wall, User Interface and Application Developer | mgwall@cyverse.org

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



 [Learning Center Home](#)

1.5 Glossary & Acronyms

A

- **action:** automate a workflow in the context of CI/CD, see [GitHub Actions](#)
- **agile:** [development methodology](#) for organizing a team to complete tasks organized over short periods called 'sprints'
- **allocation:** portion of a resource assigned to a particular recipient, typical unit is a core or node hour
- **Anaconda:** open source data science platform. Anaconda.com
- **application:** also called an 'app', a software designed to help the user to perform specific task
- **awesome:** a curated set of lists that provide insight into [awesome software projects on GitHub](#)
- **AVU:** Attribute-Value-Unit a components for [iRODS metadata](#).

B

- **beta:** β , a software version which is not yet ready for publication but is being tested
- **bash:** Bash is the GNU Project's shell, the [Bourne-Again Shell](#)
- **biocontainer:** a community-driven project that provides the infrastructure and basic guidelines to create, manage and distribute bioinformatics packages (e.g conda) and containers (e.g docker, singularity)
- **bioconda:** a channel for the conda package manager specializing in bioinformatics software

C

- **CLI:** (1) the UNIX shell [command line interface](#), most typically BASH (2) the [CyVerse Learning Institute](#)
- **command:** a set of instructions sent to the computer, typically in a typed interface

- **conda:** an installation type of the Anaconda data science platform. Command line application for managing packages and environments
- **container:** virtualization of an operating system run within an isolated user space
- **Continuous Integration:** (CI) is testing automation to check that the application is not broken whenever new commits are integrated into the main branch
- **Continuous Delivery:** (CD) is an extension of ‘continuous integration’ to make sure that you can release new changes in a sustainable way
- **Continuous Deployment:** a step further than ‘continuous delivery’, every change that passes all stages of your production pipeline is released
- **Continuous Development:** a process for iterative software development and is an umbrella over several other processes including ‘continuous integration’, ‘continuous testing’, ‘continuous delivery’ and ‘continuous deployment’
- **Continuous Testing:** a process of testing and automating software development.
- **CRAN:** The [Comprehensive R Archive Network](#)
- **CyVerse tool:** Software program that is integrated into the back end of the DE for use in DE apps
- **CyVerse app:** graphic interface of a tool made available for use in the DE

D

- **Debian:** a [free OS](#), base of other Linux distributions such as Ubuntu
- **Development:** the environment on your computer where you write code
- **DevOps** Software **Dev*elopment and information technology **Op*erations* techniques for shortening the time to change software in relation to CI/CD*
- **Discovery Environment (DE):** a data science workbench for running executable, interactive, and high throughput applications in [CyVerse DE](#)
- **distribution:** abbreviated as ‘distro’, an operating system made from a software collection based upon the Linux kernel
- **Docker:** [Docker](#) is an open source software platform to create, deploy and manage virtualized application containers on a common operating system (OS), with an ecosystem of allied tools. A program that runs and handles life-cycle of containers and images
- **DockerHub:** an official registry of docker containers, operated by Docker. [DockerHub](#)
- **DOI:** a digital object identifier. A persistent identifier number, managed by the [doi.org](#)
- **Dockerfile:** a text document that contains all the commands you would normally execute manually in order to build a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile

E

- **environment:** software that includes operating system, database system, specific tools for analysis
- **entrypoint:** In a Dockerfile, an ENTRYPOINT is an optional definition for the first part of the command to be run

F

- **FOSS:** (1) [Free and Open Source Software](#), (2) [Foundational Open Science Skills](#) - this class!
- **function:** a named section of a program that performs a specific task

G

- **git:** a version control system software
- **gitter:** a Github based messaging service that uses markdown [gitter.im](#)
- **GitHub:** a website for hosting `git` repositories – owned by Microsoft [GitHub](#)
- **GitLab:** a website for hosting `git` repositories [GitLab](#)
- **GitOps:** using `git` framework as a means of deploying infrastructure on cloud using Kubernetes
- **GPU:** graphic processing unit
- **GUI:** graphical user interface

H

- **hack:** a quick job that produces what is needed, but not well
- **HPC:** high performance computer, for large synchronous computation
- **HTC:** high throughput computer, for many parallel tasks

I

- **IaaS:** [Infrastructure as a Service](#). online services that provide APIs
- **iCommands:** [command line application](#) for accessing iRODS Data Store
- **IDE:** integrated development environment, typically a graphical interface for working with code language or packages
- **instance:** a single virtual machine
- **image:** self-contained, read-only ‘snapshot’ of your applications and packages, with all their dependencies
- **iRODS:** an open source integrated Rule-Oriented Data Management System, [iRODS.org](#)

J

- **Java:** programming language, class-based, object-oriented
- **JavaScript:** programming language
- **JSON:** Java Script Object Notation, data interchange format that uses human-readable text
- **Jupyter(Hub,Lab,Notebooks):** an IDE, originally the iPythonNotebook, operates in the browser [Project Jupyter](#)

K

- **kernel:** central component of most operating systems (OS)
- **Kubernetes:** an open source container orchestration platform created by Google [Kubernetes](#) is often referred to as `K8s`

L

- **lib:** a UNIX library
- **linux:** open source Unix-like operating system

M

- **makefile:** a file containing a set of directives used by a [make build automation tool](#)
- **markdown:** a lightweight markup language with plain text formatting syntax
- **metadata::** data about data, useful for searching and querying
- **multi-thread:** a process which runs on more than one CPU or GPU core at the same time

- **master node:** responsible for deciding what runs on all of the cluster's nodes. Can include scheduling workloads, like containerized applications, and managing the workloads' lifecycle, scaling, and upgrades. The master also manages network and storage resources for those workloads
- **Mac OS X:** Apple's popular desktop OS

N

- **node:** a computer, typically 1 or 2 core (with many threads) server in a cloud or HPC center

O

- **ontology:** formal naming and structural hierarchy used to describe data, also called a [knowledge graph](#)
- **organization:** a group, in the context of GitHub a place where developers contribute code to repositories
- **Operating System (OS):** software that manages computer hardware, software resources, and provides common services for computer programs
- **Open Science Grid (OSG):** national, distributed computing partnership for data-intensive research open-sciencegrid.org
- **ORCID:** Open Researcher and Contributor ID ([ORCID](#)), a persistent digital identifier that distinguishes you from every other researcher

P

- **PaaS:** [Platform as a Service](#) run and manage applications in cloud without complexity of developing it yourself
- **package:** an app designed for a particular language
- **package manager:** a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner
- **Production:** environment where users access the final code after all of the updates and testing
- **Python:** interpreted, high-level, general-purpose programming language [Python.org](https://python.org)

Q

- **QUAY.io:** private Docker registry [QUAY.io](https://quay.io)

R

- **R:** data science programming language [R Project](https://r-project.org)
- **recipe file:** a file with installation scripts used for building software such as containers, e.g. Dockerfile
- **registry:** a storage and content delivery system, such as that used by Docker
- **remote desktop:** a VM with a graphic user interface accessed via a browser
- **repo(sitory):** a directory structure for hosting code and data
- **RST:** ReStructuredText, a markdown type file
- **ReadTheDocs:** a web service for rendering documentation (that this website uses) readthedocs.org and readthedocs.com
- **root:** the administrative user on a linux kernel - use your powers wisely

S

- **SaaS:** [Software as a Service](#) web based platform for using software
- **schema:** a metadata standard for labeling, tagging or coding for recording & cataloging information or structuring descriptive records. see schema.org
- **scrum:** daily set of tasks and evaluations as part of a sprint.

- **shell:** is a command line interface program that runs other programs (may be complex, technical programs or very simple programs such as making a directory). These simple, stand-alone programs are called commands
- **Singularity:** a container software, used widely on HPC, created by [SyLabs](#)
- **SLACK:** Searchable Log of All Conversation and Knowledge, a team communication tool [slack.com](#)
- **sprint:** set period of time during which specific work has to be completed and made ready for review
- **Singularity def file:** (definition file) recipe for building a Singularity container
- **Stage:** environment that is as similar to the production environment as can be for final testing

T

- **tar:** software utility for collecting many files into one archive file, often referred to as a tarball
- **tensor:** algebraic object that describes a linear mapping from one set of algebraic objects to another
- **terminal:** a windowed emulator for directly entering commands to a computer
- **thread:** a CPU process or a series of linked messages in a discussion board
- **tool:** In the context of CyVerse Discovery Environment, a Docker Container
- **TPU:** tensor processing unit
- **Travis:** [Travis-CI](#), a continuous integration software

U

- **Ubuntu:** most popular [Linux OS distribution](#), based on Debian
- **UNIX:** operating system
- **user:** the profile under which applications are started and run, `root` is the most powerful system administrator

V

- **VICE:** [Visual Interactive Computing Environment](#) - Cyverse Data Science Workbench
- **virtual machine:** is a software computer that, like a physical computer, runs an operating system and applications

W

- **waterfall:** software development broken into linear sequential phases, similar to a Gantt chart
- **webGL:** JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser without the use of plug-ins
- **Windows:** Microsoft's most popular desktop OS
- **workspace:** (vs. repo)
- **worker node:** A cluster typically has one or more nodes, which are the worker machines that run your containerized applications and other workloads. Each node is managed from the master, which receives updates on each node's self-reported status.

X

- **XML:** Extensible Markup Language, data interchange format that uses human-readable text

Y

- **YAML:** [YAML Ain't Markup Language](#), data interchange format that uses human-readable text

Z

- **ZenHub:** team collaboration solution built directly into GitHub that uses kanban style boards

- **Zenodo:** general-purpose open-access repository developed under the European OpenAIRE program and operated by CERN
 - **zip:** a compressed file format
 - **zsh:** Z-Shell, now the default shell on new Mac OS X
-

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



1.6 About FOSS

Foundational Open Science Skills (FOSS) is a novel training designed to prepare principal investigators and their lab teams, both new and established, to meet the growing expectations of funding agencies, publishers, and research institutions for scientific reproducibility and data accessibility.

This training will cover key concepts about open science:

- Collaboration culture
 - Software essentials
 - Using CyVerse cyberinfrastructure
 - Data management
 - Cloud computing
-

1.6.1 Learning Objectives

By the end, participants should have the basic skills needed to:

- Work on the command line
 - Write basic scripts in Python, R, and the shell
 - Manage, share, and analyze data on CyVerse
 - Submit and run jobs in high performance and cloud computing environments
 - Communicate and collaborate on data science projects with other scientists and software developers
-

Our goals are to have each participant leave with a plan for how they will set up their own data science lab to conduct open science

- intra- and inter-lab communication and collaboration
- workflow management systems
- data management systems
- task management
- documentation

Approach

FOSS camp focuses on hands on activities – participants should learn skills by practicing them. Most of the skill taught in this course are interdependent, and lessons are designed to reinforce skills learned earlier in the week. Some material may be completely new to you, and some may be familiar. If you have experience with a subject, please help your less-experienced neighbor.

During this week, we will introduce many new tools and technologies, but these are just a sample of what is available for you to use in your own Open Science Lab. Before choosing which technologies to adopt, you should look around and see what others in your community are using. This is a case where fitting in with your peers can make you more productive. For example, if everyone you know is already on Google Chat, you may have a hard time convincing them to use Slack or Gitter, and vice versa.

We don't expect anyone to be experts at these skills after one week. Instead, we aim to give you enough hands on experience to confidently continue with your own self-paced learning. Each lesson includes links to additional learning materials that can enhance or expand on what you have learned at FOSS camp.

For up to one year following FOSS camp, CyVerse will provide concierge service to participants, to ensure that they are able to take full advantage of CyVerse infrastructure for their open science projects.

[See the Agenda](#)

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



1.7 Code of Conduct

All attendees, speakers, sponsors and volunteers at FOSS Camp are required to agree with the following code of conduct. Organisers will enforce this code throughout the event. We expect cooperation from all participants to help ensure a safe, inclusive, and collaborative environment for everybody.

Harassment by any individual will not be tolerated and may result in the individual being removed from the Camp.

Harassment includes: offensive verbal comments related to gender, gender identity and expression, age, sexual orientation, disability, physical appearance, body size, race, ethnicity, religion, technology choices, sexual images in public spaces, deliberate intimidation, stalking, following, harassing photography or recording, sustained disruption of talks or other events, inappropriate physical contact, and unwelcome sexual attention.

Workshop staff are also subject to the anti-harassment policy. In particular, staff should not use sexualised images, activities, or other material that conflicts with the code of conduct.

Participants who are asked to stop any harassing behavior are expected to comply immediately. If a participant engages in harassing behavior, the workshop organisers may take any action they deem appropriate, including warning the offender or expulsion from the workshop with no refund.

If you are being harassed, or notice that someone else is being harassed, or have any other concerns, please contact a member of the workshop staff immediately. Workshop staff will be happy to help participants contact local law enforcement, provide escorts, or otherwise assist those experiencing harassment to feel safe for the duration of the workshop. We value your attendance.

We expect participants to follow these rules at conference and workshop venues and conference-related social events.

See <http://www.ashedryden.com/blog/codes-of-conduct-101-faq> or The Carpentries https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html for more information on Codes of Conduct.

To see the Diversity & Inclusion information for CyVerse, visit <https://cyverse.org/diversity-and-inclusion>.

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



 [Learning Center Home](#)

1.8 Open Science Introductory Activity

Presentation: [What is Open Science?](#)

Open Science Introductory Activity

- Introductions in pairs.

- Describe your work or research area and draw a picture of the other person's research.

1.8.1 Planning your own Open Science lab

- Download a copy of the [template](#)
- Rename the file replacing “template” with your name.

Day 1

Question - Communication

Knowing your work style, how do you think you will be most productive in a team?

Answer

[Communication](#)

Question - Collaboration

How might you use a team charter in your work?

Answer

Create a Charter that has:

- Start and End Dates
 - Project Description
 - Roles and Responsibilities
 - Project Justification
 - List of Deliverables
 - Approach
 - Success Criteria
 - Key Assumptions
-
-

Discussion - Project Management

Imagine you are managing a medium-sized project with collaborators in three different locations. Describe how you would use technologies like GitHub, Gitter, Slack, Wikis, or Jira to manage your project.

Hints

[Version Control with GitHub Documentation](#)

Day 2

Question - Project Management

How could you combine R and GitHub to manage your project?

Answer

[Using Git & RStudio](#)

Discussion - Project Management

Describe how you could use R to make your research more reproducible. Include specific examples from a project you are working on.

Discussion - Data Management

List two specific ways (i.e. related to an actual project you are working on) in which you could use command line skills to improve your productivity.

Day 3

Question - Collaboration :class: admonition-question

Which CyVerse features would make collaboration easier for you?

Answer

[CyVerse Features](#)

Discussion - Cloud Computing

Describe a situation in which you would use cloud computing or HPC to scale your research?

How would you go about setting up the solution?

Hint

[Use Free Research Cyberinfrastructure](#)

Day 4

Question - Data Management

How would you use the CyVerse Data Store to manage the data for a complex, distributed project?

Answer

CyVerse Data Store

Question - Data Management

What is some element of a data management plan that you had not thought of before today? How would you make a plan for this element?

Answer

[Data Management Overview](#)

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



 [Learning Center Home](#)

1.9 Collaboration Culture and Roles

Instructor: [Michael Mandel](#), Eller College of Business

[Mike's Personal Website](#)

[Michael's Powerpoint Presentation-Monday](#) (link to download this file)

[Michael's Powerpoint Presentation-Friday](#) (link to download this file)

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



1.10 Command Line and the Unix Shell

1.10.1 Setup

Launch Atmosphere Instance

1. Go to <https://atmo.cyverse.org/> and click 'Projects' at the top of the page.
2. Click the 'create new project' button and enter a name and description.
3. Click 'images' at the top of the screen.
4. Select the image called 'Ubuntu 18.04 GUI XFCE Base'
5. Click launch.
6. For this work you can leave all the settings as default and click 'launch instance'.

Once Instance is 'Active'

1. Click on the image name 'Ubuntu 18.04 GUI XFCE Base' in your project.
2. Click 'Open Web Desktop' on the bottom right corner of the screen.
3. On web desktop accept 'default config'.

Get Some Data

1. Click the globe icon at the bottom of the Desktop. This will open FireFox.
2. Copy this link <http://swcarpentry.github.io/shell-novice/data/data-shell.zip> to the FireFox search bar to download the data. Choose 'save file'.
3. To move the file to your (web) Desktop open the file manager (folder icon on Desktop). Open the downloads folder. Drag the 'data-shell.zip' file onto the Desktop.
4. Unzip/extract the file: Right click the file and select 'extract here'. You should end up with a new folder called 'data-shell' on your Desktop.
5. Open a terminal by selecting the command line icon at the bottom of the desktop.
6. In the terminal, type cd and hit enter.

1.10.2 Background

At a high level, computers do four things:

- run programs
- store data

- communicate with each other, and
- interact with us

The graphical user interface (GUI) is the most widely used way to interact with personal computers.

- give instructions (to run a program, to copy a file, to create a new folder/directory) with mouse
- intuitive and very easy to learn
- scales very poorly

The shell - a command-line interface (CLI) to make repetitive tasks automatic and fast.

- can take a single instruction and repeat it

Example

If we have to copy the third line of each of a thousand text files stored in thousand different folders/directories and paste it into a single file line by line.

- Using the traditional GUI approach will take several hours to do this.
- Using the shell this will only take a couple of minutes (at most).

The Shell

The Shell is a program which runs other programs rather than doing calculations itself.

- programs can be as complicated as a climate modeling software
- as simple as a program that creates a new folder/directory
- **simple programs used to perform stand alone tasks are usually referred to as commands.**
- most popular Unix shell is **Bash**, (the Bourne Again SHell).
- Bash is the default shell on most modern implementations of Unix

To see which shell you are using

```
$ echo $SHELL
/bin/bash
```

A typical shell window looks something like:

```
bash-3.2$
bash-3.2$ ls -F /
Applications/      System/
Library/           Users/
Network/           Volumes/
bash-3.2$
```

first line shows only a prompt

- indicates the shell is waiting for input
- your shell may use different text for the prompt

- **do not type the prompt**, only the commands that follow it

the second line

- command is ls, with an option -F and an argument /
- options change the behavior of a command
- each part is separated by spaces
- capitalization matters
- commands can have more than one option or argument
- commands don't always require an option or argument

lines 3-5 contain output that command produced

- this is a list of files and folders in the root directory (/)

Finally, the shell again prints the prompt and waits for you to type the next command.

Warning: Spaces and capitalization are important!

The command line is always case sensitive.

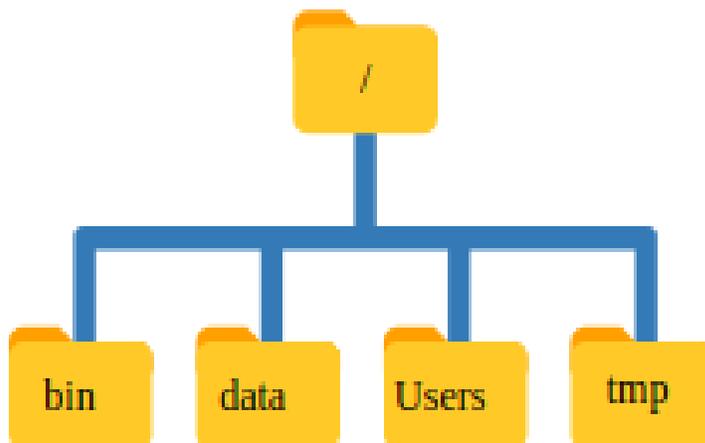
There is always a space between command and option.

Hint: To re-enter the same command again use the up arrow to display the previous command. Press the up arrow twice to show the command before that (and so on).

1.10.3 Navigating Files and Directories

File System

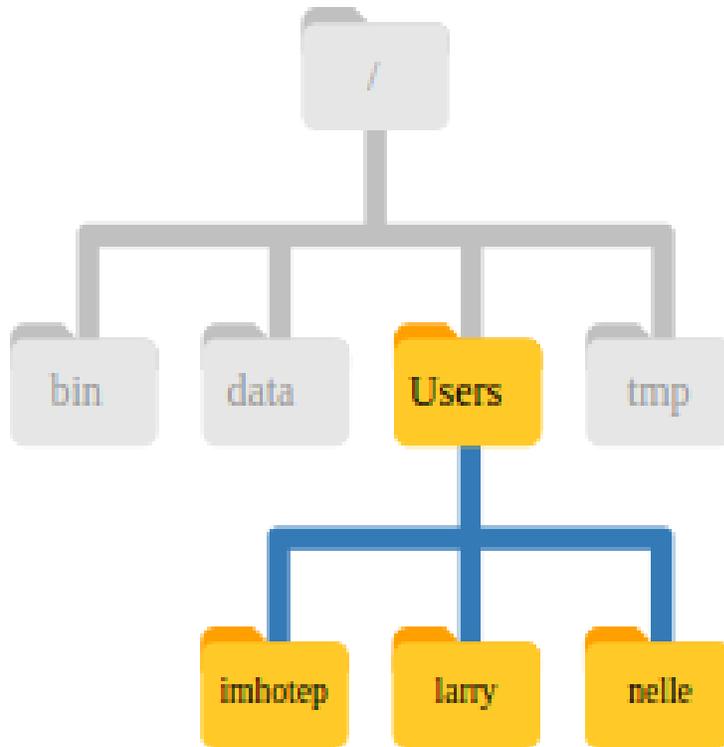
files—hold information **directories** (also called “folders”)—hold files or other directories



At the top is the **root directory that holds everything else**. We refer to it using a slash character, /, on its own; this is the leading slash in /home/amcooksey.

Inside that directory are other directories.

Underneath /Users, we find one directory for each user with an account on Nelle's machine, her colleagues imhotep and larry.



Note: Directory names will vary depending on operating system.

This is just an example. Today we are all using Ubuntu Linux. Instead of 'Users' you will have a directory called 'home'. It works the same way.

Print working directory (pwd)

Directories are like places - at any time while we are using the shell we are in exactly one place, called our current working directory. **Commands mostly read and write files in the current working directory**, i.e. "here", so knowing where you are before running a command is important. **pwd shows you where you are:**

```
$ pwd
/home/amcooksey
```

Typically, when you open a new command prompt you will be in your home directory to start.

List files and directories (ls)

We can see the contents of our current directory by running ls.

```
$ ls
Desktop Documents  Downloads  Music      Pictures   Public    Templates Videos
```

Your results may be slightly different depending on your operating system and how you have customized your filesystem.

ls prints the names of the files and directories in the current directory. We can make its output more comprehensible by using the **option -F** (also known as a switch or an option), which tells ls to add a marker to file and directory names to indicate what they are. A trailing `/` indicates that this is a directory.

```
$ ls -F
Desktop/  Documents/  Downloads/  Music/  Pictures/  Public/  Templates/  Videos/
```

Here, we can see that our home directory contains mostly **sub-directories**. Any names in your output that don't have trailing slashes, are plain old **files**.

`ls -l` displays the items in a list with more information:

```
$ ls -l
total 32
drwxr-xr-x 2 amcooksey root          4096 Jan 29 10:02 Desktop
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Jan 29 10:02 Documents
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Jan 29 10:02 Downloads
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Jan 29 10:02 Music
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Jan 29 10:02 Pictures
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Jan 29 10:02 Public
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Jan 29 10:02 Templates
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Jan 29 10:02 Videos
```

Change directory (cd)

We can change our location to a different directory, so we are no longer located in our home directory. The command doesn't change the directory, it changes the shell's idea of what directory we are in.

Hint: Begin typing a file or directory and press **Tab**. The shell will autocomplete the name.

We can move into the directory we downloaded to our Desktop:

```
$ cd Desktop
```

We can see what's there using `ls`:

```
$ cd data
$ ls -l
total 116
-rw-r--r-- 1 amcooksey iplant-everyone 283 Aug 7 20:13 amino-acids.txt
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Aug 7 20:14 animal-counts
-rw-r--r-- 1 amcooksey iplant-everyone 136 Aug 7 20:13 animals.txt
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Aug 7 20:13 elements
-rw-r--r-- 1 amcooksey iplant-everyone 554 Aug 7 20:13 morse.txt
drwxr-xr-x 2 amcooksey iplant-everyone 4096 Aug 7 20:14 pdb
-rw-r--r-- 1 amcooksey iplant-everyone 8898 Aug 7 20:13 planets.txt
-rw-r--r-- 1 amcooksey iplant-everyone 45 Aug 7 20:13 salmon.txt
-rw-r--r-- 1 amcooksey iplant-everyone 73861 Aug 7 20:13 sunspot.txt
```

We now know how to go down the directory tree, but how do we go up?

With our methods so far, `cd` can only see sub-directories inside your current directory. There are different ways to see directories above your current location; we'll start with the simplest.

cd without an argument will return you to your home directory

```
$ cd
$ pwd
/home/amcooksey
```

The shell interprets the character ~ (**tilde**) at the start of a path to mean “**the current user’s home directory**”. So cd ~ will also take you to your home directory.

```
$ cd ~
$ pwd
/home/amcooksey
```

- (**dash**) character. cd will translate - into **the previous directory I was in**, which is faster than having to remember, then type, the full path.

move up one directory level that looks like this:

```
$ cd ..
```

.. is a special directory name meaning “the directory containing this one”

To move up two directory levels:

```
$ cd ../..
```

In the same way that .. represents the directory above the current working directory, . represents the current working directory. More on this later.

Relative vs Absolute Paths

When you use a **relative path** with a command like ls or cd, it tries to find that **location from where we are**, rather than from the root of the file system.

However, it is possible to specify the **absolute path** to a directory by including its **entire path from the root directory, which is indicated by a leading slash**. The leading / tells the computer to follow the path from the root of the file system, so it always refers to exactly one directory, no matter where we are when we run the command.

1.10.4 Working with Files and Directories

Creating directories (mkdir)

As you might guess from its name, **mkdir means “make directory”**. Make a new directory called thesis.

```
$ mkdir thesis
```

Since thesis is a relative path (i.e., does not have a leading slash, like /what/ever/thesis), the new directory is created in the current working directory:

```
$ ls -F
creatures/  data/  molecules/  north-pacific-gyre/  notes.txt  pizza.cfg
solar.pdf  thesis/  writing/
```

Good Names for Files and Directories

Complicated names of files and directories can make your life painful when working on the command line. Here we provide a few useful tips for the names of your files.

1. Don't use spaces.

Spaces can make a name more meaningful, but since spaces are used to separate arguments on the command line it is better to avoid them in names of files and directories. You can use - or _ instead (e.g. north-pacific-gyre/ rather than north pacific gyre/).

2. Don't begin the name with - (dash).

Commands treat names starting with - as options.

3. Stick with letters, numbers, . (period or 'full stop'), - (dash) and _ (underscore).

If you need to refer to names of files or directories that have spaces or other special characters, you should surround the name in quotes ("").

Since we've just created the thesis directory, there's nothing in it yet:

```
$ ls -F thesis
```

Creating a text files

With a text editor

Let's change our working directory to thesis using `cd`, then run a **text editor called Nano** to create a file called `draft.txt`:

```
$ cd thesis
$ nano draft.txt
```

Let's type in a few lines of text. Once we're happy with our text, we can press **Ctrl+O** (press the Ctrl or Control key and, while holding it down, press the O key) to write our data to disk (we'll be asked what file we want to save this to: press **Return** to accept the suggested default of `draft.txt`).

```
GNU nano 2.0.6 File: draft.txt Modified
It's not "publish or perish" any more,
it's "share and thrive".
█
```

```
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Once our file is saved, we can use **Ctrl-X** to **quit** the editor and return to the shell.

In nano, along the bottom of the screen you'll see `^G Get Help ^O WriteOut`. This means that you can use Control-G to get help and Control-O to save your file.

nano doesn't leave any output on the screen after it exits, but `ls` now shows that we have created a file called `draft.txt`:

```
$ ls
draft.txt
```

With touch

We have seen how to create text files using the nano editor. Now, try the following command:

```
$ touch my_file.txt
```

What did the touch command do?

Use `ls -l` to inspect the files. How large is `my_file.txt`?

```
$ ls -l
```

Note: You may have noticed that files are named “something dot something”, and in this part of the lesson, we always used the extension `.txt`. This is just a convention: we can call a file `mythesis` or almost anything else we want. However, most people use two-part names most of the time to help them (and their programs) tell different kinds of files apart. The second part of such a name is called the filename extension, and indicates what type of data the file holds.

Naming a PNG image of a whale as `whale.mp3` doesn’t somehow magically turn it into a recording of whalesong, though it might cause the operating system to try to open it with a music player when someone double-clicks it.

Moving files and directories (`mv`)

Returning to the `data-shell` directory,

```
$ cd ~/Desktop/data-shell/
```

In our `thesis` directory we have a file `draft.txt` which isn’t a particularly informative name, so let’s change the file’s name using **`mv`**, which is short for “move”:

```
$ mv thesis/draft.txt thesis/quotes.txt
```

The **first argument tells `mv` what we’re “moving”**, while the **second is where it’s to go**. In this case, we’re moving `thesis/draft.txt` to `thesis/quotes.txt`, which has the **same effect as renaming the file**. Sure enough, `ls` shows us that `thesis` now contains one file called `quotes.txt`:

```
$ ls thesis
quotes.txt
```

Warning: One has to be careful when specifying the target file name, since **`mv` will silently overwrite any existing file with the same name**, which could lead to data loss. An additional option, **`mv -i` (or `mv -interactive`)**, can be used to make `mv` ask you for confirmation before overwriting.

`mv` also works on directories

Let’s move `quotes.txt` into the current working directory. We use `mv` once again, but this time we’ll just use the name of a directory as the second argument to tell `mv` that we want to keep the filename, but put the file somewhere new. (This is why the command is called “move”.) In this case, the directory name we use is the special directory name `.` that we mentioned earlier.

```
$ mv thesis/quotes.txt .
```

The effect is to move the file from the directory it was in to the current working directory. `ls` now shows us that `thesis` is empty:

```
$ ls thesis
```

Copying Files and Directories (`cp`)

The `cp` command works **very much like `mv`**, **except it copies** a file instead of moving it. We can check that it did the right thing using `ls` with two paths as arguments — like most Unix commands, `ls` can be given multiple paths at once:

```
$ cp quotes.txt thesis/quotations.txt
$ ls quotes.txt thesis/quotations.txt
quotes.txt  thesis/quotations.txt
```

We can also **copy a directory** and all its contents by using the **recursive option `-r`**, e.g. to back up a directory:

```
$ cp -r thesis thesis_backup
```

We can check the result by listing the contents of both the `thesis` and `thesis_backup` directory:

```
$ ls thesis thesis_backup
thesis:
quotations.txt

thesis_backup:
quotations.txt
```

Removing files and directories (`rm`)

Returning to the `data-shell` directory, let's tidy up this directory by removing the `quotes.txt` file we created. The Unix command we'll use for this is **`rm` (short for 'remove')**:

```
$ rm quotes.txt
```

We can confirm the file has gone using `ls`:

```
$ ls quotes.txt
ls: cannot access 'quotes.txt': No such file or directory
```

`rm` can remove a directory and all its contents if we use the recursive option `-r`, and it will do so without any confirmation prompts:

```
$ rm -r thesis
```

Warning: Deleting Is Forever

The Unix shell doesn't have a trash bin that we can recover deleted files from. Instead, when we delete files, they are unlinked from the file system so that their storage space on disk can be recycled. Given that there is no way to retrieve files deleted using the shell, `rm -r` should be used with great caution (**you might consider adding the interactive option `rm -r -i`**).

1.10.5 Operations with multiple files and directories

Oftentimes one needs to copy or move several files at once. This can be done by providing a list of individual filenames, or specifying a naming pattern using wildcards.

Copy with Multiple Filenames

For this exercise, you can test the commands in the data-shell/data directory.

In the example below, what does cp do when given several filenames and a directory name?

```
$ mkdir backup
$ cp amino-acids.txt animals.txt backup/
```

If given more than one file name followed by a directory name (i.e. the destination directory must be the last argument), cp copies the files to the named directory.

Using wildcards for accessing multiple files at once

***** is a wildcard, which matches zero or more characters. Let's consider the data-shell/molecules directory: *.pdb matches ethane.pdb, propane.pdb, and every file that ends with '.pdb'. On the other hand, p*.pdb only matches pentane.pdb and propane.pdb, because the 'p' at the front only matches filenames that begin with the letter 'p'.

? is also a wildcard, but it matches exactly one character. So ?ethane.pdb would match methane.pdb whereas *ethane.pdb matches both ethane.pdb, and methane.pdb.

Wildcards can be used in combination with each other e.g. ???ane.pdb matches three characters followed by ane.pdb, giving cubane.pdb ethane.pdb octane.pdb.

1.10.6 Other Useful Tools and Commands

sudo

allows users to run programs with the security privileges of the superuser

This command is used as a prefix to other commands that you need elevated permissions to run. Which commands you will need this for will vary depending on the computer you are using at the time. If you receive a **permission denied** error it is likely that you will need 'sudo' to run the command.

```
$ docker run hello-world:latest
```

```
$ sudo docker run hello-world:latest
```

Note: Use 'sudo' with caution. Sometimes important files restrict permission because they are very sensitive and it is un-wise to change them unless you know what you are doing.

head

prints the first few (10 by default) lines of a file

```
$ head data/sunspot.txt
(* Sunspot data collected by Robin McQuinn from *)
(* http://sidc.oma.be/html/sunspot.html *)
```

(continues on next page)

(continued from previous page)

```
(* Month: 1749 01 *) 58
(* Month: 1749 02 *) 63
(* Month: 1749 03 *) 70
(* Month: 1749 04 *) 56
(* Month: 1749 05 *) 85
(* Month: 1749 06 *) 84
(* Month: 1749 07 *) 95
```

tail

prints the last few (10 by default) lines of a file

```
$ tail data/sunspot.txt
(* Month: 2004 05 *) 42
(* Month: 2004 06 *) 43
(* Month: 2004 07 *) 51
(* Month: 2004 08 *) 41
(* Month: 2004 09 *) 28
(* Month: 2004 10 *) 48
(* Month: 2004 11 *) 44
(* Month: 2004 12 *) 18
(* Month: 2005 01 *) 31
(* Month: 2005 02 *) 29
```

history

displays the last few hundred commands that have been executed

```
$ history
1988 cd ..
1989 ls
1990 cd data-shell/
1991 ls
1992 mkdir thesis
1993 ls
1994 ls-F
1995 ls
1996 cd Desktop/data-shell/data/
1997 pwd
1998 cd ..
1999 pwd
2000 ls -F
2001 cd Desktop/data-shell/
2002 head data/sunspot.txt
2003 tail data/sunspot.txt
2004 history
```

grep

finds and prints lines in files that match a pattern

```
$ cd
$ cd Desktop/data-shell/writing
$ cat haiku.txt
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

```
$ grep "not" haiku.txt
Is not the true Tao, until
"My Thesis" not found
Today it is not working
```

find

finds files

To find all the files in the 'writing' directory and sub-directories

```
$ find .
.
./thesis
./thesis/empty-draft.md
./tools
./tools/format
./tools/old
./tools/old/oldtool
./tools/stats
./haiku.txt
./data
./data/two.txt
./data/one.txt
./data/LittleWomen.txt
```

To find all the files that end with '.txt'

```
$ find -name *.txt
./haiku.txt
```

echo

print strings (text)

This is especially useful when writing Bash scripts

```
$ echo hello world
hello world
```

>

prints output to a file rather than the shell

```
$ grep not haiku.txt > not_haiku.txt
$ ls
data haiku.txt not_haiku.txt thesis tools
```

>>

appends output to the end of a file

```
$ grep Tao haiku.txt >> not_haiku.txt
$ nano not_haiku.txt
```

lnano>>|

|

directs output from the first command into the second command (and the second into the third)

```
$ cd ../north-pacific-gyre/2012-07-03
$ wc -l *.txt | sort -n | head -n 5
240 NENE02018B.txt
300 NENE01729A.txt
300 NENE01729B.txt
300 NENE01736A.txt
300 NENE01751A.txt
```

wget

downloads things from the internet

```
$ cd ~/Desktop
$ rm data-shell.zip
$ wget http://swcarpentry.github.io/shell-novice/data/data-shell.zip
```

Getting help and further learning

Note: This is was just a brief summary of how to use the command line. There is much, much more you can do. For more information check out the [Software Caprentry](#) page.

There are two common ways to find out how to use a command and what options it accepts:

The help option

We can pass a `-help` option to the command, such as:

```
$ ls --help
```

The man command

The other way to learn about `ls` is to type

```
$ man ls
```

This will open the manual in your terminal with a description of the `ls` command and its options and, if you're lucky, some examples of how to use it.

To navigate through the man pages, you may use `↑` and `↓` to move line-by-line, or try **B** and **Spacebar** to skip up and down by a full page.

To quit the man pages, press `q`.

Manual pages on the web

Of course there is a third way to access help for commands: searching the internet via your web browser. When using internet search, including the phrase `unix man page` in your search query will help to find relevant results. GNU provides links to its [manuals](#) including the [core GNU utilities](#), which covers many commands introduced within this lesson.

- On Github: [|Github Repo Link|](#)
- Send feedback: Tutorials@CyVerse.org



1.11 Basics of Linux

Modern web, cloud, high performance computing, and most data science applications are run on operating systems (OS) other than Microsoft Windows. To do data intensive science, you need a familiarity with **Linux**. We've scheduled several sections during FOSS for working on Linux Systems, including on the [Command Line Interface \(CLI\)](#) and [Unix Shell](#), and using CyVerse' [Atmosphere Cloud](#), which runs Linux OS virtual machines.

The good news comes in two parts. First, whether you know it or not, you **probably** already use Linux or a platform based on Linux, on a daily basis. *Do you have an Android or iOS phone?* If you own a [Mac OS X](#) device, you already enjoy many of the benefits of a Linux-like OS, including access to a terminal. Second, the Linux experience has generally been described as [satisfying](#), and many users report moving on from Windows OS to Linux comes [without regret](#).

Over [87%](#) of the personal computer market still relies on the popular Microsoft OS. However, the landscape changes completely for mobile apps ([99%](#) Linux or Linux-like [Android, iOS], [<0.1%](#) Windows), web ([66%](#) Linux, [32%](#) Windows), and cloud or HPC ([100%](#) Linux). Microsoft is acutely aware of this disparity, and is actively working to integrate Linux into their OS, including their [acquisition of GitHub](#) (and [how it has changed](#)), and the release of [Windows Subsystem for Linux \(WSL\) 2](#).

1.11.1 Common Linux Operating Systems

The most common operating systems you'll see used for data science are:

- [Alpine](#) - small and lightweight, useful in container applications
- [CentOS](#) - stable, reliable, most commonly used on web and cloud servers
- [Debian](#) - lightweight, utilitarian, stable
- [Ubuntu](#) - utilitarian, user friendly, most popular distribution, based on Debian

Enterprise Distributions:

- [Red Hat](#) - based on open source software, you pay for customer support

1.11.2 *Installing Linux*

Desktop-based Distributions

- [Ubuntu](#)
- [Debian](#)
- [Mint](#) - “modern, elegant and comfortable operating system which is both powerful and easy to use.”
- [OpenSUSE](#) - “The makers’ choice for sysadmins, developers and desktop users.”

Windows Subsystem for Linux

The so-called “WSL” is a complete linux subsystem that runs under Windows 10. Microsoft recently announced [WSL 2.0](#).

Windows Linux Dual boot

Not ready to take the Linux plunge yet? Why not set up a Windows-Linux dual boot?

- [Ubuntu](#)
- [Mint](#)

Package Managers

Linux uses [package management](#) services to install programs. If you’re a R user, this should seem familiar.

Packages can be installed on the command line, or in graphic UI.

1.11.3 *Self Paced*

[Best Linux Distributions for Beginners](#)

[Beginners Guide to Linux](#)

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



1.12 Introduction to Reproducible Science

The so-called reproducibility crisis (see , ,) is something you have probably heard about (and maybe one of the reasons you have come to FOSS). Headlines in the media (such as) definitely give pause to researchers and ordinary citizens who hope that the science used to recommend a course of medical treatment or design self-driving cars is sound.

Before we go further, it's actually important to ask what is reproducibility?

Question

How do you define reproducible science?

Answer

In , Hans Plesser gives the following useful definitions:

- **Repeatability** (Same team, same experimental setup): The measurement can be obtained with stated precision by the same team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same location on multiple trials. For computational experiments, this means that a researcher can reliably repeat her own computation.
- **Replicability** (Different team, same experimental setup): The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts.
- **Reproducibility** (Different team, different experimental setup): The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently.

The paper goes on to further simplify:

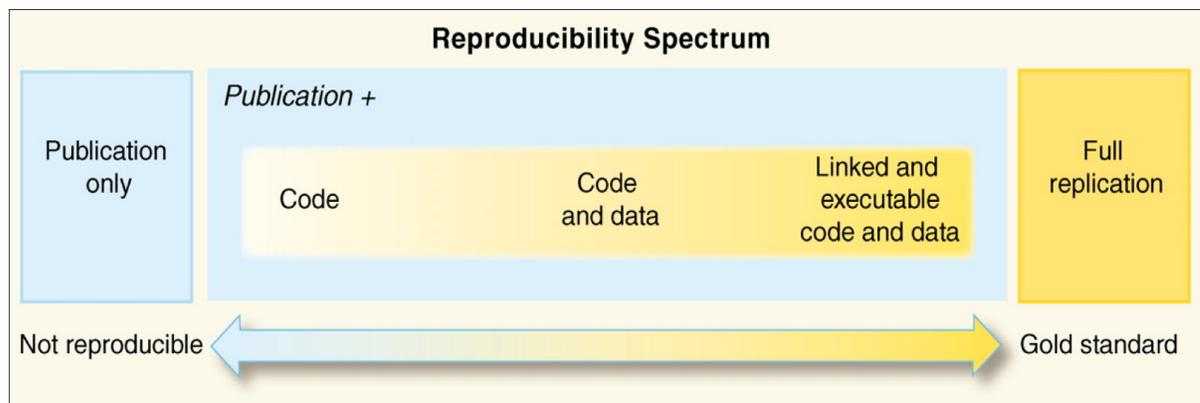
- **Methods reproducibility**: provide sufficient detail about procedures and data so that the same procedures could be exactly repeated.
- **Results reproducibility**: obtain the same results from an independent study with procedures as closely matched to the original study as possible.
- **Inferential reproducibility**: draw the same conclusions from either an independent replication of a study or a reanalysis of the original study.

Discussion Question

How do these definitions apply to your research/teaching?

Work with your fellow learners to develop a shortlist of ways reproducibility relates to your work. Try to identify challenges and even successes you'd like to share.

Often, when we say “reproducibility” we mean all or at least several of the concepts the proceeding discussion encompasses. Really, reproducibility can be thought of as set values such as some laboratories express in a code of conduct, (see for example or). Reproducibility comes from our obligations and desires to work ethically, honestly, and with confidence that the data and knowledge we produce is done has integrity. Reproducibility is also a “spectrum of practices”, not a single step. (See figure below from).



Assuming you have taken in the potentially anxiety inducing information above, the most important thing to know is that there is a lot of help to make reproducibility a foundation of all of your research.

1.12.1 Reproducibility Tutorial

In the following tutorial, we will introduce some of the software introduced at FOSS in the context of creating a reproducible experiment. The goal will be to give an example of how connecting various software pieces and practices can generate a well-documented research project. Though the example will be drawn from a biological example, the ideas an approaches apply to the sciences in general.

-

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



1.13 Communication

We will cover both internal and external communication strategies during the workshop.

1.13.1 Internal

Choosing which software to use for your internal lab communication can be complicated by the cost of setting up, the cost of maintaining, and simply by the sheer number of platforms that are out there.

For this workshop, we use **SLACK** (Searchable Log of All Conversation & Knowledge). Microsoft's competitor to SLACK is **Microsoft Teams**.

Remember, the intention of these platforms are to **improve productivity** & not become a distraction.

Other popular alternatives

- BaseCamp
- Discord
- Mastodon
- Mattermost

Create a SLACK Workspace

1. Create a new Workspace
2. Create channels, add apps & tools

Create a Gitter

1. Create a Workspace
2. Create channels

1.13.2 External

Although we didn't cover it explicitly in the announcement for the workshop, communicating with the public and other members of your science community is one of the most important parts of your science!

There are many ways scientists use social media and the web to share their data science ideas:

1. “Science Twitter” - is really just regular [Twitter](#), but with a focus on following other scientists and organizations, and tweeting about research you're interested in. By building up a significant following, more people will know you, know about your work, and you'll have a higher likelihood of meeting other new collaborators.
2. [Blogs](#) - there are numerous platforms for blogging about research, the older platforms tend to dominate this space. Other platforms like, [Medium](#) offer a place for researchers to create personalized reading spaces and self publish.
3. Community groups - There are lists (and lists of lists) of [nationals research organizations](#), in which a researcher can become involved. These older organizations still rely on official websites, science journal blogs, and email lists to communicate with their members. In the earth sciences there are open groups which focus on communication like the [Earth Science Information Partners \(ESIP\)](#) with progressive ideas about how data and science can be done. Other groups, like [The Carpentries](#) and [Research Bazaar](#) are focused on data science training and digital literacy.

Important: Remember Personal and Professional Accounts are Not Isolated

You decide what you post on the internet. Your scientist identity may be a part of your personal identity on social media, it might be separate. A future employer or current employer can see your old posts. What you post in your personal accounts can be considered a reflection of the organization you work for and may be used in decisions about hiring or dismissal.

1.13.3 Self-Paced Material

[15 Data Science Communities to Join](#)

[Python & Slack](#)

[Slack CLI notifications](#)

[Gitter Services](#)

[Meetups](#)

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
 - Live chat/help: Click on the  [reproducible_science/./img/intercom.png](#) on the bottom-right of the page for questions on documentation
-



1.14 GitHub

Version Control is an important component in software development which manages changes to documents, websites, code, or other digital information. Version control can save you when code changes break things. Web hosting of your code repositories lets you share and work on code together and save your work in the event of a hardware failure.

The most common version control software in data science are `git`, `svn`, `cvs`, and `bzr`.

Given the limited amount of time we have this week, we are only going to cover `Git` and one web-based hosting service (`GitHub`) in this camp.

In this lesson you will learn how to:

- **Interact with `git` via `GitHub`**
 - Using a browser
 - Using the command line
- Add collaborators to your organization
- **Initialize a `git` repository on the command line and push it to `GitHub`**
 - Push/pull files
 - Commit to `GitHub`
- **Learn about versions**
 - Forks
 - Branches
 - Merges
 - Revert
 - Releases
 - Badging

Important: `GitHub` is not really intended for storing or manipulating data

1.14.1 Navigating & Interacting with `GitHub`

Basic Layout



The basic layout includes:

- Profile
- Organizations
- Teams
- Repositories
- [Projects](#)
- Followers
- Contributions
- Watch / Unwatch repos

Add a Collaborator

Online only

1. Go to your Organization
2. Click “people”
3. Invite someone!

OR

1. Can add collaborators under “Settings” -> “Collaborators”

Create a Repo Online & Locally

Online

1. Go to your dashboard
2. Click “New”
3. Name the repository
4. Initialize with “README” - form of metadata - this tells you and the those looking at the repository what it is about
5. Go to “Clone or Download” and clone to local folder / directory titled “Lab”
6. Can delete repo under “Settings” -> “Options”

Command Line

1. Locate or create a local folder / directory structure titled “Lab”

```
cd
mkdir -p github/lab
cd github/lab
```

2. Initialize folder as a repository with git

```
git init
echo "This is a test repo" >> README.md #append to file
git add . #add all the files in the current directory
git commit -m "initial project"
```

Note: Documentation is vital *You will be doing this for YOU 6 months from now*

Interact with GitHub

Clone a repo

This is used to work *locally* rather than online.

Online

1. Click the down arrow “Clone or download”
2. Click “Open in Desktop”
3. Select where to save it - Create a folder for GitHub repos on your computer locally

Command Line

```
cd path/to/location
git clone <url> [rename]
#URL of thte repository on GitHub
#rename the directory (optional)
```

Commit

Online

1. Create a file by clicking “Create a new file”
2. Name file (/name)
3. Write commit message
4. Press “Commit”

OR

1. Create a file *locally*
2. Click “Upload files”
3. Select file(s) within a folder

Command Line

1. Create a file

```
cd path/to/repo
touch file.txt
```

2. Add file(s)

```
git add -A #adds all the new files
git push
git commit -m "added file" #-m initiates a message
```

Tip: It is good practice to write commit messages to remember what you added or fixed.

Create an Issue

Issues are great for tracking decisions made or to-do lists

Online only

1. Click on the repository you just created.
2. Click on “Issues”
3. Click on “New issue”
4. Create a title (# and issue number for reference)
5. Assign to someone, or create a label
6. Submit new issue
7. Close issue

Version Control

Online

1. Click on file
2. Click “History”

Command Line

```
git log
git log --stat #gives abbreviated stats for each commit
git log --pretty=oneline #can also add options: short, full, fuller
git log --pretty=format:"%h - %an, %ar : %s" #lots of options for pretty=format
q #to quit
```

Create a branch

Branches are useful to working on code etc. without disturbing the *master branch*.

Online

1. Select the down arrow on the repository page that says “Branch:master”
2. Create a new branch name

OR

1. Add new file or edit existing file.
2. Write a commit message.
3. Select “Create a **new branch**”

Command Line

1. Create a new branch

```
cd path/to/repo
#common practice to pull before committing anything
git pull #does a fetch for you
git checkout -b new-branch #creates a new branch and puts you on that branch
#set new branch upstream
git push --set-upstream origin new-branch
```

2. Edit some files

```
#edit files
vi path/to/file
q #to quit
```

3. Commit changes

```
git pull
git add
commit -am "changed a file" #stage changes and write a message
git push
```

Make a pull request

Pull requests are useful to have another set of eyes to review changes before merging them with the master branch.

Online Only

1. From your branch, create a new file
2. Commit file to your branch
3. Hit “Compare & pull request”
4. Go to pull requests
5. Set it on the master branch

Merge

Online

1. Go to “Pull requests”
2. Select down arrow of “Merge pull request” (if no conflicts)
3. Delete branch
4. Leave comment if need be
5. Close pull request

Command Line

```
cd path/to/repo
git pull
git checkout new-branch
git merge master #testing to see if merging breaks anythin
git checkout master
git merge new-branch #now repull it all into master
```

Revert

Command Line

```
git log
#copy tag for last working commit
git revert <tag to last working commit>
#will make it look like a new commit
git add -A
git commit -m "changed things back to <commit tag>"
git push
```

Advanced

Fork

Online

1. Go to a new repository
2. Click fork
3. Save to personal repository.
4. Clone to Desktop.
5. Interact via online or in Desktop.
6. If want to make suggestions, can create a pull request.

Command Line

```
git clone <github-repo>
cd <new-folder>
git fork
```

Etc.

These are *Online only*

- **Versioning**

1. Go to “Releases”
2. Click “Create a new release”
3. Tag version: Version #
4. Release title: I usually put the date of the release, but any system can work

- **Reactions:**

1. Create a new pull request
2. Looking at the messages, click the smiley face to give a reaction

- **Badges**

1. Go find a [badge!](#)
2. Copy badge code into README:

Git cheat sheet

Here is a list of the most important commands in Git:

Git Task	Command	Description
Set up your profile locally	<code>git config --global user.name "Cy Unicorn"</code>	Set your user name
	<code>git config --global user.email Cyl@cyverse.org</code>	Set your email address
Create a Repository locally	<code>git init</code>	Initialize a folder as a git repository
Get an existing repository from a web service	<code>git clone ssh://git@github.com/[username]/[repository-name].git</code>	Create a local copy of a remote repository

Branching & Merging	Description
<code>git branch</code>	List branches (the asterisk denotes the current branch)
<code>git branch -a</code>	List all branches (local and remote)
<code>git branch [branch name]</code>	Create a new branch
<code>git branch -d [branch name]</code>	Delete a branch
<code>git push origin --delete [branch name]</code>	Delete a remote branch
<code>git checkout -b [branch name]</code>	Create a new branch and switch to it
<code>git checkout -b [branch name] origin/[branch name]</code>	Clone a remote branch and switch to it
<code>git checkout [branch name]</code>	Switch to a branch
<code>git checkout -</code>	Switch to the branch last checked out
<code>git checkout -- [file-name.txt]</code>	Discard changes to a file
<code>git merge [branch name]</code>	Merge a branch into the active branch
<code>git merge [source branch] [target branch]</code>	Merge a branch into a target branch
<code>git stash</code>	Stash changes in a dirty working directory
<code>git stash clear</code>	Remove all stashed entries

Sharing & Updating Projects	Description
<code>git push origin [branch name]</code>	Push a branch to your remote repository
<code>git push -u origin [branch name]</code>	Push changes to remote repository (and remember the branch)
<code>git push</code>	Push changes to remote repository (remembered branch)
<code>git push origin --delete [branch name]</code>	Delete a remote branch
<code>git pull</code>	Update local repository to the newest commit
<code>git pull origin [branch name]</code>	Pull changes from remote repository
<code>git remote add origin ssh://git@github.com/[username]/[repository-name].git</code>	Add a remote repository
<code>git remote set-url origin ssh://git@github.com/[username]/[repository-name].git</code>	Set a repository's origin branch to SSH

Inspection & Comparison	Description
<code>git log</code>	View changes
<code>git log --summary</code>	View changes (detailed)
<code>git diff [source branch] [target branch]</code>	Preview changes before merging



1.15 CI/CD

Building your own software, data management or workflow management systems requires you to do a significant amount of interpersonal management, as well as tracking of development.

Software engineers having long suffered under the burden of disorganization and communication with clients, have come up with a framework for developing software and sharing it with their users.

While data science applications have a different audience and intended result, the organizational practices of software developers are a valuable and useful tool to consider integrating into your open science lab group.

Frequently Used Terms

- **Continuous Integration:** (CI) is testing automation to check that the application is not broken whenever new commits are integrated into the main branch
- **Continuous Delivery:** (CD) is an extension of ‘continuous integration’ to make sure that you can release new changes in a sustainable way
- **Continuous Deployment:** a step further than ‘continuous delivery’, every change that passes all stages of your production pipeline is released
- **Continuous Development:** a process for iterative software development and is an umbrella over several other processes including ‘continuous integration’, ‘continuous testing’, ‘continuous delivery’ and ‘continuous deployment’
- **Continuous Testing:** a process of testing and automating software development.
- **Development:** the environment on your computer where you write code
- **DevOps:** Development and information technology Operations is the set of practices surrounding CI/CD
- **Production:** environment where users access the final code after all of the updates and testing
- **Stage:** environment that is as similar to the production environment as can be for final testing

1.15.1 Continuous Development

The software developer concept of ‘continuous delivery’ can be applied to your data science projects and lab.

As we've discussed, version control is an important component of modern software development. Critically, version control can also be used in data science applications and for research project management. There are two dominant forms of project management for continuous delivery in open source software: [Waterfall](#) and [Agile Scrum](#).

<https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>

Agile

Agile development practices involve organizing a team around short term (1-2 week long) 'sprints'. Sprints are organized by [scrum master](#). Team members are assigned tasks and evaluate their results during sprint reviews and planning sessions.

Waterfall

Similar to the common [Gantt chart](#) a [waterfall model](#) is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialisation of tasks.

Note: In this workshop, we're working with GitHub, but there are other services, like [GitLab](#) or [Bitbucket](#) which might fit your needs better.

1.15.2 Continuous Integration

Doing reproducible science requires you to host your code and versioned software used to complete the analysis, in addition to the actual data. GitHub or Gitlab could become the central point supporting your data science lab.

Powerful uses of GitHub include integration with other web services, like container registries ([DockerHub](#)), websites ([ReadTheDocs](#), web sites <https://pages.github.com/>), continuous integration ([CircleCI](#), [Jenkins](#), [Travis](#)), and workflow managers [GitHub Actions](#).

[Continuous Integration \(CI\)](#) is a practice of checking code repositories (typically a few times a day) to test for changes which may cause failures.

CI can be integrated into either scientific programming workflows or into code development

The most popular CI tools are:

- [Travis CI](#) - fast, easy to set up, cloud based
- [Circle CI](#) - fast, easy to set up, cloud based
- [Jenkins](#) - free, can be hosted internally (requires server), highly customizable (plugins)

When to use CI?

- building or hosting services to a community
- developing versioned copies of containers for public consumption
- DevOps + Data Science

Travis CI

Setup

Circle CI

Setup

Jenkins

Jenkins is a bit harder to set up because you need a dedicated server

Setup

GitHub Actions

GitHub now offers 'actions' which serve as an [integrated CI](#) for your repositories

1.15.3 Badges

Status badges can be embedded in a README.md. Badges let you show the state of code or documentation.

You can view a diverse list of different badges on [Shields.io](#)

Now you can pass the `style` GET argument, to get custom styled badges same as you would for shields.io. If no argument is passed, `flat` is used as default.

STYLE	BADGE
flat	
flat-square	
for-the-badge	
plastic	
social	

1.15.4 Self paced

Circle vs Jenkins vs Travis

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



1.16 Websites & Documentation

This website is rendered using a [ReadTheDocs.org](#) template. Think of RTD as “Continuous Documentation”.

ReadTheDocs has become a popular tool for developing web-based documentation. [CyVerse Learning Materials Github](#) hosts a few templates which you can view and pull for your own use.

[Bookdown](#) is an open-source R package that facilitates writing books and long-form articles/reports with R Markdown.

[Github Pages](#) using [Jekyll](#) or [Bootstrap.js](#), are another popular way of hosting websites via [Github](#).

[Confluence Wikis \(CyVerse\)](#) are another tool for documenting your workflow.

1.16.1 *Some things to remember*

ReadTheDocs

- publishing websites via [ReadTheDocs.com](#) costs money.
- You can work in an offline state, where you develop the materials and publish them to your localhost using [Sphinx](#)
- You can work on a website template in a [Github](#) repository, and pushes are updated in near real time using [ReadTheDocs.com](#).

Bookdown

- Bookdown websites can be hosted by [RStudio Connect](#)
- You can publish a Bookdown website using [Github Pages](#)

Github Pages

- You can pull templates from other [Github](#) users for your website, e.g. [jekyll themes](#)
- [Github](#) pages are free, fast, and easy to build, but limited in use of subdomain or URLs.

Confluence Wiki

- CyVerse is paying for [Confluence](#) access.
- Integration with other platforms like [Jira](#).
- Scripting in the [Confluence Wiki](#) is different than other platforms and may be less reproducible.

1.16.2 Build your own Website

ReadTheDocs

1. [Install](#)
2. [Use Github](#)
3. [Create a ReadTheDocs account](#)

Bookdown

1. Install R and RStudio
2. Install Bookdown package

```
install.packages("bookdown", dependencies=TRUE)
```

3. Open the Bookdown demo and get started

GitHub Pages

1. Create a GitHub account
2. Clone the repo `https://github.com/username/username.github.io`
3. Create an `index.html`
4. Push it back to GitHub

CyVerse Confluence Wiki

1. Create a CyVerse Account and log into `https://wiki.cyverse.org`
2. Create a a new Blank Page
3. Set permissions to share or make private

1.16.3 Methodology

Protocols.io - collaborative platform and preprint server for: science methods, computational workflows, clinical trials, operational procedures, safety checklists, and instructions / manuals.

QUBES - community of math and biology educators who share resources and methods for preparing students to tackle real, complex, biological problems.

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



1.17 Data Management Overview

1.17.1 Why should you care about data management?

If you give your data to a colleague who has not been involved with your project, will they be able to make sense of it? Will they be able to use it effectively and properly?

If you come back to your own data in five years, will you be able to make sense of it? Will you be able to use it effectively and properly?

When you are ready to publish a paper, is it easy to find all the correct versions of all the data you used and present them in a comprehensible manner?

Data management produces self-describing datasets that: - make life much easier for you and your collaborators - benefit the scientific research community by allowing others to reuse your data - are required by most funders and many journals - *Recent Dear Colleague letter from NSF* - NSF proposal preparation guidelines: https://www.nsf.gov/pubs/policydocs/pappg19_1/pappg_11.jsp#XID4

1.17.2 Data Management Basics

The Data Life Cycle

Data management is the set of practices that allow researchers to effectively and efficiently handle data throughout the data life cycle. Although typically shown as a circle (below) the actual life cycle of any data item may follow a different path, with branches and internal loops. Being aware of your data's future helps you plan how to best manage them.



Image from Strasser et al..

Data Types

Different types of data require different management practices. What are some of the considerations for managing:

- tabular data
 - images
 - sound or video recordings
 - geospatial data
 - sensor data
 - raw versus processed data
 - files versus datasets
-

Best practices for the data life cycle

The summary below is adapted from the excellent [DataONE best practices primer](#).

Plan:

- Describe the data that will be compiled, and how the data will be managed and made accessible throughout its lifetime.
- A good plan considers each of the stages below.

Collect:

- Have a plan for data organization in place before collecting data.
- Collect and store observation metadata at the same time you collect the metadata.
- Take advantage of machine generated metadata.

Assure:

- Record any conditions during collection that might affect the quality of the data.
- Distinguish estimated values from measured values.
- Double check any data entered by hand.
- Perform statistical and graphical summaries (e.g., max/min, average, range) to check for questionable or impossible values.
- Mark data quality, outliers, missing values, etc.

Describe: Comprehensive data documentation (i.e. metadata) is the key to future understanding of data. Without a thorough description of the context of the data, the context in which they were collected, the measurements that were made, and the quality of the data, it is unlikely that the data can be easily discovered, understood, or effectively used.

- Organize your data for publication. Before you can describe your data, you must decide how to organize them. This should be planned before hand, so that data organization is a minimal task at the time of publication.
- **Thoroughly describe**
 - the dataset (e.g., name of dataset, list of files, date(s) created or modified, related datasets)
 - **the people and organizations involved in data collection (e.g., authors, affiliations, sponsor)**
 - * Go get an [ORCID](#) if you don't have one.
 - the scientific context (reason for collecting the data, how they were collected, equipment and software used to generate the data, conditions during data collection, spatial and temporal resolution)
 - **the data themselves**
 - * how each measurement was produced
 - * units
 - * format
 - * quality assurance activities
 - * precision, accuracy, and uncertainty

Metadata standards and ontologies are invaluable for supporting data reuse.

- **Metadata standards tell you**
 - which metadata attributes to include
 - how to format your metadata

- what values are allowable for different attributes
- **Some metadata standards**
 - [DataCite](#) (for publishing data)
 - [Dublin Core](#) (for sharing data on the web)
 - Minimum Information for any (x) Sequence ([MIxS](#))
 - [OGC standards](#) for geospatial data
- **Ontologies provide standardization for metadata values**
 - Example: [Environment Ontology](#) terms for the MIxS standards
 - Example: [Plant Ontology](#) for plant tissue types or development stages
- [FAIRSharing.org](#) lists standards and ontologies for Life Sciences.

The CyVerse Data Commons supports good data description through

- Metadata templates (remember the [DataCite](#) metadata template from yesterday)
- Bulk metadata upload ([example dataset](#))
- Automatic collection of analysis parameters, inputs, and outputs in the DE.

Preserve:

- **To be FAIR data must be preserved in an appropriate long-term archive (i.e. data center).**
 - Sequence data should go to INSDC (usually [NCBI](#))
- Identify data with value – it may not be necessary to preserve all data from a project
- The [CyVerse Data Commons](#) provides a place to publish and preserve data that was generated on or can be used in CyVerse, where no other repository exists.
- See lists of repositories at [FAIRSharing.org](#).
- Github repos can get DOIs through [Zenodo](#).
- **Be aware of licensing and other intellectual property issues**
 - See licensing information below
 - Repositories will require some kind of license, often the least restrictive
 - Repositories are unlikely to enforce reuse restrictions, even if you apply them.

Discover:

- Good metadata allows you to discover your own data!
- **Databases, repositories, and search indices provide ways to discover relevant data for reuse**
 - <https://toolbox.google.com/datasetsearch>
 - <https://www.dataone.org/>
 - [FAIRSharing.org](#) lists databases for Life Sciences.

Integrate:

- Data integration is a lot of work.
- Standards and ontologies are key to future data integration
- **Know the data before you integrate them**

- Don't trust that two columns with the same header are the same data
- **Properly cite the data you reuse!**
 - Use DOIs wherever possible

Analyze:

- Follow open science principles for reproducible analyses (CyVerse, RStudio, notebooks, IDEs)
 - State your hypotheses and analysis workflow before collecting data. Tools like [OSF](#) allow you to make this public.
 - Record all software, parameters, inputs, etc.
-

FAIR data

How to make data findable, accessible, interoperable, and reusable.

See the [FAIR data page](#).

Open versus Public versus FAIR:

One definition of open: <http://opendefinition.org/>

FAIR does not demand that data be open.

1.17.3 References and Resources

[DataOne best practices](#)

[Center for Open Science](#)

Licensing information:

- *Open Data Commons* <<https://opendatacommons.org/licenses/index.html>>-
 - [Creative Commons](#)
 - – <https://choosealicense.com/>
-

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



1.18 FAIR Data

Vocabulary

- Metadata
- PID: Permanent Identifier
- GUID: Globally Unique Identifier
- Ontology
- Provenance
- License
- Search Index
- Access protocol

1.18.1 FAIR Principles

In 2016, the [FAIR Guiding Principles](#) for scientific data management and stewardship were published in Scientific Data. Read it. Its short.

Findable

- F1. (meta)data are assigned a globally unique and persistent identifier
- F2. data are described with rich metadata (defined by R1 below)
- F3. metadata clearly and explicitly include the identifier of the data it describes
- F4. (meta)data are registered or indexed in a searchable resource

Accessible

- A1. (meta)data are retrievable by their identifier using a standardized communications protocol
- A1.1 the protocol is open, free, and universally implementable
- A1.2 the protocol allows for an authentication and authorization procedure, where necessary
- A2. metadata are accessible, even when the data are no longer available

Interoperable

- I1. (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.
- I2. (meta)data use vocabularies that follow FAIR principles
- I3. (meta)data include qualified references to other (meta)data

Reusable

- R1. meta(data) are richly described with a plurality of accurate and relevant attributes
- R1.1. (meta)data are released with a clear and accessible data usage license
- R1.2. (meta)data are associated with detailed provenance
- R1.3. (meta)data meet domain-relevant community standard

1.18.2 CARE Principles

The for Indigenous Data Governance were drafted at the International Data Week and Research Data Alliance Plenary co-hosted event “Indigenous Data Sovereignty Principles for the Governance of Indigenous Data Workshop,” 8 November 2018, Gaborone, Botswana.

Collective Benefit

- C1. For inclusive development and innovation
- C2. For improved governance and citizen engagement
- C3. For equitable outcomes

Authority to Control

- A1. Recognizing rights and interests
- A2. Data for governance
- A3. Governance of data

Responsibility

- R1. For positive relationships
- R2. For expanding capability and capacity
- R3. For Indigenous languages and worldviews

Ethics

- E1. For minimizing harm and maximizing benefit
- E2. For justice
- E3. For future use

1.18.3 FAIR - TLC

Traceable, Licensed, and Connected

- The need for metrics: <https://zenodo.org/record/203295#.XkrzTxNKjzI>

Question

Which do you think is the hardest, F, A, I, or R, and why?

Question

What is the best way to cite data?

Question

What are the relative values of a data publication versus a peer-reviewed paper?

Question

What role do ontologies play in FAIR-TLC?

1.18.4 Hands on Exercise - Metadata in CyVerse

- Using metadata in the DE
 - iCommands metadata: `imeta`
 - example 1: `imeta add -C B123 test_metadata value unit`
 - example 2: `imeta add -d B123/0123.txt test_metadata value unit`
-

1.18.5 References and Resources

<https://www.nature.com/articles/sdata201618>

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



 Learning Center Home

1.19 Data Management Plans (DMP)

“A data management plan or DMP is a formal document that outlines how data are to be handled both during a research project, and after the project is completed.[1] The goal of a data management plan is to consider the many aspects of data management, metadata generation, data preservation, and analysis before the project begins; this may lead to data being well-managed in the present, and prepared for preservation in the future.”

(Source: https://en.wikipedia.org/wiki/Data_management_plan)

Example DMP

Why bother?

Stick: You have to make one

Reviewers definitely look at them, but they may not be enforced.

Carrot: Make your life easier

- Planning for you project makes it run more smoothly
- Avoid surprise costs

1.19.1 Elements of a good DMP

- **Information about data & data format(s)**
 - data types
 - data sources
 - analysis methods
 - formats
 - QA/QC
 - version control
 - **data life cycle**
- **Metadata content and format(s)**
 - format
 - standards
- **Policies for access, sharing, and re-use**
 - funder obligations
 - ethical and privacy issues (data justice)
 - intellectual property, copyright, citation
 - timeline for releases
- **Long-term storage, data management, and preservation**
 - which data to preserve
 - which archive/repository
- **Budget (PAPPG)**
 - each of the above elements cost time/money
 - Personnel time for data preparation, management, documentation, and preservation (including time)
 - Hardware and/or software for data management, back up, security, documentation, and preservation (including time)
 - Publication/archiving costs (including time)

Not only what, but *who* (roles).

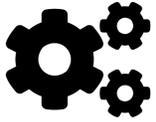
Extra challenges for collaborative projects.

1.19.2 Machine actionable DMPs

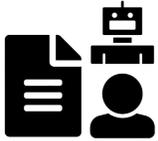
- DMPs describe research methods that will evolve over the course of a project
- to be a useful tool for researchers and others, the content must be updated to capture the methods that are employed and the data that are produced



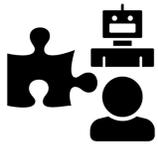
1 Integrate DMPs with the workflows of all stakeholders in the research data ecosystem



2 Allow automated systems to act on behalf of stakeholders



3 Make policies (also) for machines, not just for people



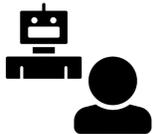
4 Describe—for both machines and humans—the components of the data management ecosystem



5 Use PIDs and controlled vocabularies



6 Follow a common data model for maDMPs



7 Make DMPs available for human and machine consumption



8 Support data management evaluation and monitoring



9 Make DMPs updatable, living, versioned documents



10 Make DMPs publicly available

(Source: <https://doi.org/10.1371/journal.pcbi.1006750.g002>)

1.19.3 Tools for DMPs

<https://dmptool.org/>

Exercise: Log in to DMP tools and create a mock DMP.

1.19.4 References and Resources

- NSF Guidelines on DMPs
 - https://dmptool.org/general_guidance
 - https://dmptool.org/public_templates
 - Professional and scholarly societies, e.g., the Ecological Society of America <http://www.esa.org/esa/science/data-sharing/resources-and-tools/>
 - DataOne - <https://www.dataone.org/best-practices>
 - Data Carpentry - <http://datacarpentry.org/>
 - The US Geological Survey <http://www.usgs.gov/datamanagement/index.php>
 - Repository registry (and search) service: <http://www.re3data.org/>
 - Your university library
-

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



 Learning Center Home

1.20 Data Management Tools

- KNB tools
- Open Science Framework
- CyVerse Data Commons

1.20.1 CyVerse Data Commons

- Metadata capability (covered earlier - see [Using metadata in the DE](#))
 - [Managing a shared project on CyVerse](#)
 - [Community Released Data Folders](#)
 - [DOIs for datasets](#)
-

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



1.21 About CyVerse

CyVerse Vision: Transforming science through data-driven discovery.

CyVerse Mission: Design, deploy, and expand a national cyberinfrastructure for life sciences research and train scientists in its use.

CyVerse provides life scientists with powerful computational infrastructure to handle huge datasets and complex analyses, thus enabling data-driven discovery. Our powerful extensible platforms provide data storage, bioinformatics tools, image analyses, cloud services, APIs, and more.

Originally created as the iPlant Collaborative to serve U.S. plant science communities, the cyberinfrastructure we have built is germane to all life sciences disciplines and works equally well on data from plants, animals, or microbes. Thus, iPlant was renamed CyVerse to reflect the broader community now served by our infrastructure. By democratizing access to supercomputing capabilities, we provide a crucial resource to enable scientists to find solutions for the future. CyVerse is of, by, and for the community, and community-driven needs shape our mission. We rely on your feedback to provide the infrastructure you need most to advance your science, development, and educational agenda.

CyVerse Homepage: <http://www.cyverse.org>

Evolution of CyVerse



CyVerse is an NSF-funded project. The project began in 2008 as ‘iPlant’ with the mission of ‘empowering a new plant biology’. Funding was renewed in 2013 for another 5 years with the new mission of ‘cyberinfrastructure for life sciences’. In 2016 the name of the project was changed from ‘iPlant’ to ‘CyVerse’ to reflect its role in all life sciences, not just plants. In 2018 CyVerse was renewed for another 5 years with our current mission: ‘to design, deploy, and expand a national Cyberinfrastructure for Life Sciences research, and to train scientists in its use’.

Over the past 10 years CyVerse priorities have focused on genomics and transcriptomics tools that were needed to deal with the huge increase in high-throughput sequencing data. While that is still a priority, CyVerse has since expanded to include image and geospatial analysis tools. CyVerse is built for data.

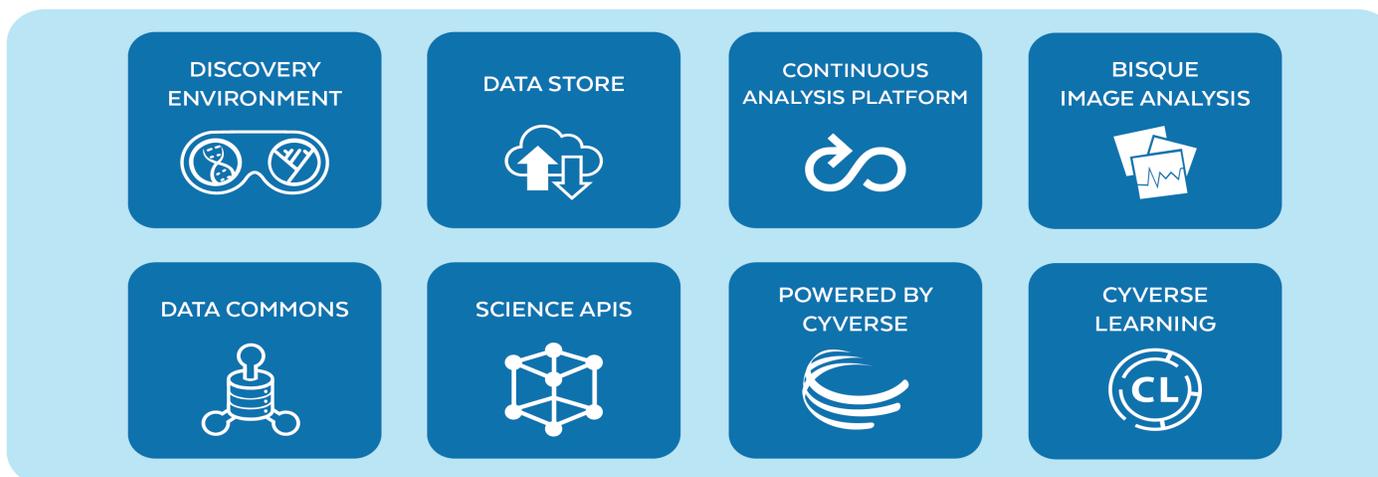
1.21.1 What is Cyberinfrastructure?

Cyberinfrastructure is a combination of

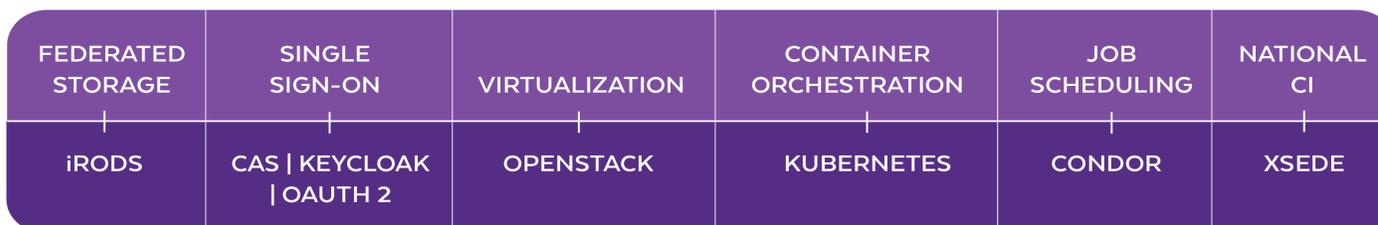
- platforms, tools and datasets researchers need to do their work
- storage and compute hardware necessary for modern analyses
- people who provide training and support

The CyVerse cyberinfrastructure can be thought of in layers. The bottom layer (on which everything else is built) consists of the hardware resources. On top of that are the services necessary to make a functional system. The next layer represents extensible services, or those parts of the system that may be adopted and used by third parties. Most users will interact primarily with the top layer which represents the various analysis and distribution platforms. While the bottom layers are the most flexible, the top layers are the most user-friendly.

PRODUCTS



SERVICES



HARDWARE RESOURCES



1.21.2 User Portal

The CyVerse user portal allows users to manage their accounts, subscriptions and events in a single place. Some things you can do here include:

Create and manage your CyVerse account

- Reset your password
- Add an email address to your account
- Change your name or username
- Change your institution, department, position
- Change your CyVerse subscriptions

Manage access to CyVerse platforms/services

- Some CyVerse services (such as Atmosphere) have additional restrictions and access must be ‘turned on’.

Manage workshops you’ve attended or hosted

Access ‘Powered by CyVerse’ projects

User portal forms

- Request a Data Store allocation increase
- Request a community released data folder
- Request a workshop or webinar
- Reserve Atmosphere cloud resources for workshops or classes
- Request an External Collaborative Partnership (ECP)
- Get Powered by CyVerse

1.21.3 Data Store



Securely store data for active analyses or sharing with your collaborators.

- **Upload, download and share your data**
 - DE simple upload/download. Convenient but not good for large files.
 - Cyberduck is a third-party software with graphic interface for transferring data. Available for Mac and Windows.
 - iCommands is more powerful/flexible, good for large transfers but requires some command line knowledge
- Data limit of 100 GB (can request increase up to 10 TB)
- Data storage is integrated into the Discovery Environment (where analyses are run).
- Share your data with collaborators
- [Data Store guide](#)

1.21.4 Discovery Environment



Use hundreds of bioinformatics apps and manage data in a simple web interface.

- Provides graphic interface for bioinformatics tools for scientists with no command line experience

- **User extensible. Users can add their own tools and make their own apps.**
 - Share them with collaborators
 - Publish them
- VICE (Visual and Interactive Computing Environment) for interactive use of Jupyter notebooks, RStudio and RShiny.
- Integrated with the Data Store for ease of use
- Share your analyses with your collaborators
- [DE guide](#)
- [VICE documentation](#)

1.21.5 Atmosphere



Create a custom cloud-based scientific analysis platform or use a ready-made one for your area of scientific interest.

- Cloud computing for life sciences
- 100s of pre-built images
- Fully customize your software setup
- Choose (or build) an image that best suits your needs
- [Atmosphere guide](#)

1.21.6 Bisque



Bio-Image Semantic Query User Environment for the exchange and exploration of image data

- Exchange, explore, and analyze biological images and their metadata.
- Image data analysis and management
- [Bisque manual](#)

1.21.7 DNA Subway



SUBWAY

Teach classroom-friendly bioinformatics for genome analysis, DNA Barcoding, and RNA-Sequencing.

- Educational tool
- **Ties together key bioinformatics tools and databases to**
 - assemble gene models
 - investigate genomes
 - work with phylogenetic trees
 - analyze DNA barcodes
- Analyze your own data or the sample data provided
- DNA Subway guide

1.21.8 Data Commons



The Data Commons provides services to manage, organize, preserve, publish, discover, and reuse data.

- Access discoverable and reusable data with metadata features and functions
- Browse Community Released Data and data curated by CyVerse
- Easily publish data to the NCBI or directly to the CyVerse Data Commons

1.21.9 Science APIs



Programmatic access to CyVerse services

- Science-as-a-service platform
- Define your own compute, and storage resources (local and CyVerse)
- Build your own app store of scientific codes and workflows
- Agave API for HPC

- Terrain API for DE

1.21.10 SciApps

A cloud-based platform for building and sharing reproducible bioinformatics workflows across distributed computing and storage systems

- Build branching analysis workflows
- SciApps guide

1.21.11 Powered by CyVerse

Powered by



CYVERSE™

Third-party projects can leverage the CyVerse cyberinfrastructure components to provide services to their users.

Some ‘Powered by CyVerse’ projects you may be familiar with

- CoGe
- BioExtract Server
- CIPRES
- ClearedLeavesDB
- Digital Imaging of Root Traits (DIRT)
- Federated Plant Database Initiative for Legumes (LegFed)
- Galaxy
- Genomes to Fields
- iMicrobe
- Integrated Breeding Platform
- SoyKB
- TERRA-REF
- TNRS- Taxonomic Name Resolution Service

1.21.12 The CyVerse Learning Center



The CyVerse Learning center is a beta release of our learning materials in the popular “Read the Docs” formatting.

- We are transitioning our learning materials into this format to make them easier to search, use, and update.

1.21.13 The CyVerse Wiki

This collaborative documentation site is used to record important information about CyVerse, its products and services, and community collaborators and their projects.

- Anyone with a CyVerse account is welcome to help out.
- User have their own spaces and can add content
- **Much of the CyVerse documentation has been moved the Learning center but some things will continue to be in the Wiki**
 - DE app documentation
 - Many tutorials

1.21.14 Intercom



Intercom is our live-chat user support app. You will find the Intercom ‘smiley’ logo in the bottom right corner of the Discovery Environment, Atmosphere, the Wiki and the user portal.

Funding and Citations

CyVerse is funded entirely by the National Science Foundation under Award Numbers DBI-0735191, DBI-1265383 and DBI-1743442.

Please cite CyVerse appropriately when you make use of our resources, [CyVerse citation policy](#)

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



 [Learning Center Home](#)

1.22 Accessing Data Store



The Data Store is more than a place to save your files – it is a way to manage the life cycle of your data. From creation to publication to beyond, there are a number of practices to ensure that the integrity and value of your data are maintained.

In this lesson we will practice other ways of accessing the Data Store, plus how to make your data publicly available.

1.22.1 Discovery Environment Interface

DE interface allows uploading and downloading one file at a time. It works best for transfer of small files < 2GB.

From the DE interface, go to “Upload -> Simple Upload from Desktop” to upload a file. For downloading a file go to “Download -> Simple Download”.

```
Sample data for practice  
/iplant/home/shared/iplantcollaborative/example_data/FOSS_2020/foss_sampledata
```

1.22.2 iCommands

iCommands is a collection of tools developed by the [iRODS project](#), which is the technology that supports the CyVerse Data Store. Using iCommands is the most flexible way to interact with the Data Store.

iCommands provides command line access to the Data Store, so it can be included in scripts to automate data upload and download. Unfortunately, the latest iCommands cannot be installed on most Windows operating systems, but participants with Windows computers can do this exercise using Atmosphere (which will be covered in tomorrow’s lessons). If you are running Windows 10, you can [run iCommands on the Linux subsystem](#).

Follow along with the [Using iCommands quick start](#) to:

- Install and configure iCommands
- Upload a file to your home folder (iput)
- Download a file to your desktop (iget)

In addition, we will use iCommands to:

- Create a new folder in your home directory (mkdir)

```
$ mkdir newdir
```

- Move a file from your home directory to the new folder (imv)

```
$ imv file_name newdir/file_name
```

- Navigate to a public folder (icd)

```
$ icd /iplant/home/shared/imicrobe/camera
```

- List files in a directory

```
$ ils
```

- Copy a public file to the newly created folder (icp)

```
$ icp camera_projects/CAM_PROJ_AcidMine.csv /iplant/home/$username/newdir/CAM_PROJ_
↪AcidMine.csv
$ icd /iplant/home/$username/
$ils newdir
$ils -A newdir
```

Here is the [full documentation of iCommands](#).

1.22.3 CyberDuck

Cyberduck is a free 3rd party software tool that allows you to drag-and-drop files between your local computer (or a remote server) and the Data Store. Cyberduck can also be used to rename files, and browse other shared or public Data Store locations.

Follow along with the [CyberDuck](#) quick start to:

- Install and configure CyberDuck
- Upload a file to your CyVerse home directory
- Navigate to a public folder

1.22.4 WebDAV

WebDAV is an extension to the HTTP protocol that allows users to remotely edit and manage files. CyVerse has added support for WebDAV to the Data Store. This means users can access their home and public folders in the CyVerse Data Store from their local computers using web browsers and other WebDAV enabled applications such as common operating system file managers. With WebDAV, users can copy files between local computer and the Data Store as easily as if they were copying them between two folders on their computer.

Follow along with the [WebDAV](#) quick start to access CyVerse data via WebDAV services.

1.22.5 Which method to choose?

Criteria	Cyberduck	iCommands	DE Interface	DE WebDAV
Ease of use	2	4	1	3
Setup required?	Yes	Yes	No	No
Works best for?	Multiple small files	Large files	Small files < 2GB	Small files < 2GB
GUI support?	Yes	No	Yes	Yes
Command-line support	Yes	Yes	No	Yes
Allows to open/edit files?	No	No	Yes	Yes

1.22.6 CyVerse Data Commons

The Data Commons provides services throughout CyVerse to manage, organize, preserve, publish, discover, and reuse data.

Data Publication

Through the Data Commons, you can submit data directly to NCBI's SRA or WGS, or request a Digital Object Identifier (DOI) for your dataset.

For data that are not stable or permanent, you can request a Community Released Folder.

For an overview see [Publishing data on the CyVerse Data Commons](#).

1.22.7 Additional Resources

[Data Store Manual](#)

[Create a public link via the DE](#)

[DOI request quick start](#)

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



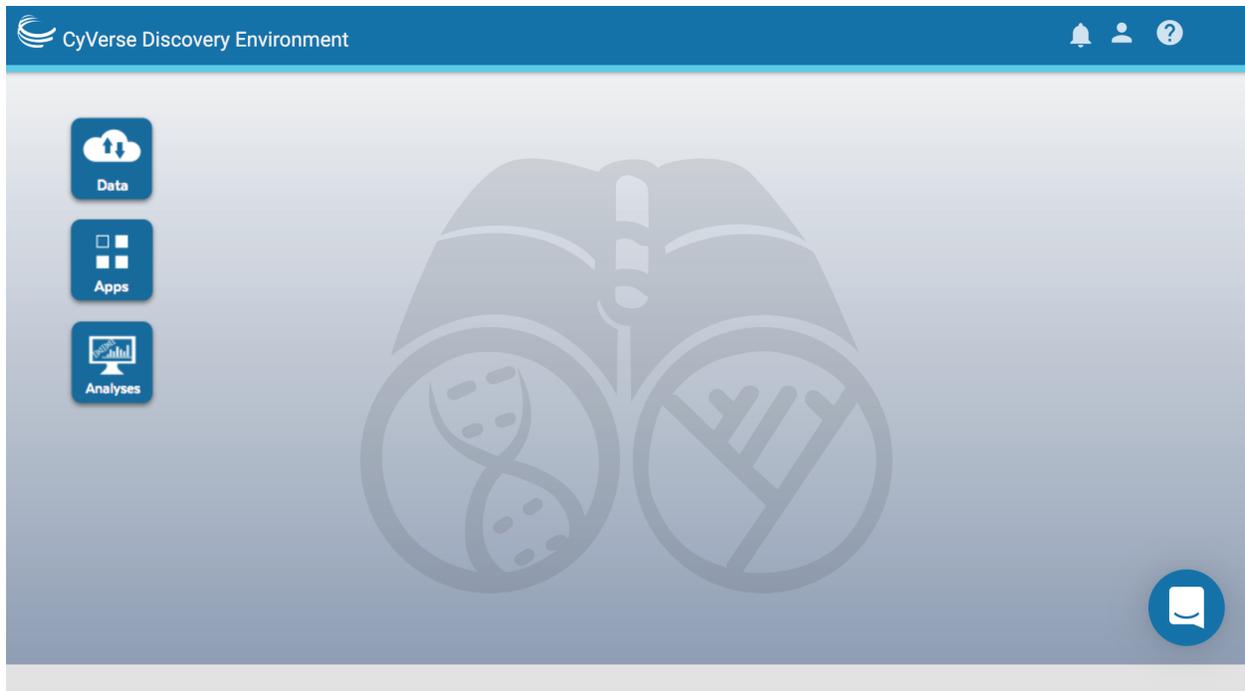
 [Learning Center Home](#)

1.23 Discovery Environment - Data Management



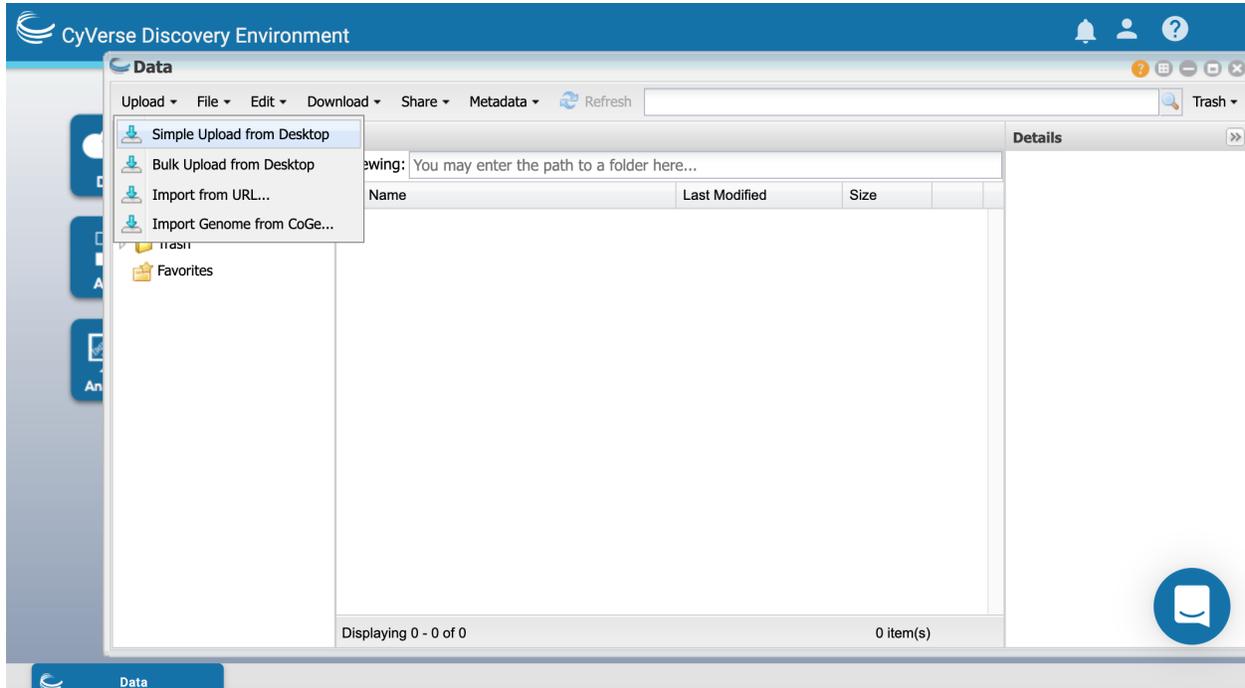
1.23.1 DE Basics Walkthrough

- Log in at <https://de.cyverse.org/>

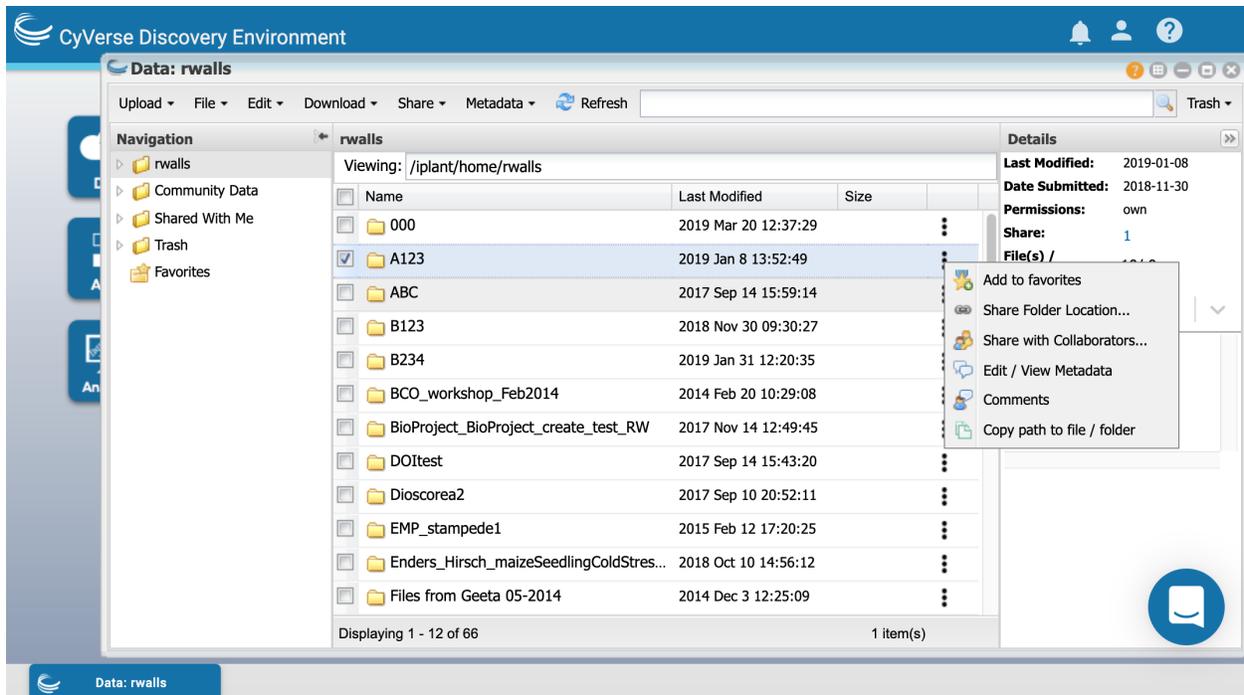


Data Window

- Open the data window and upload a file:



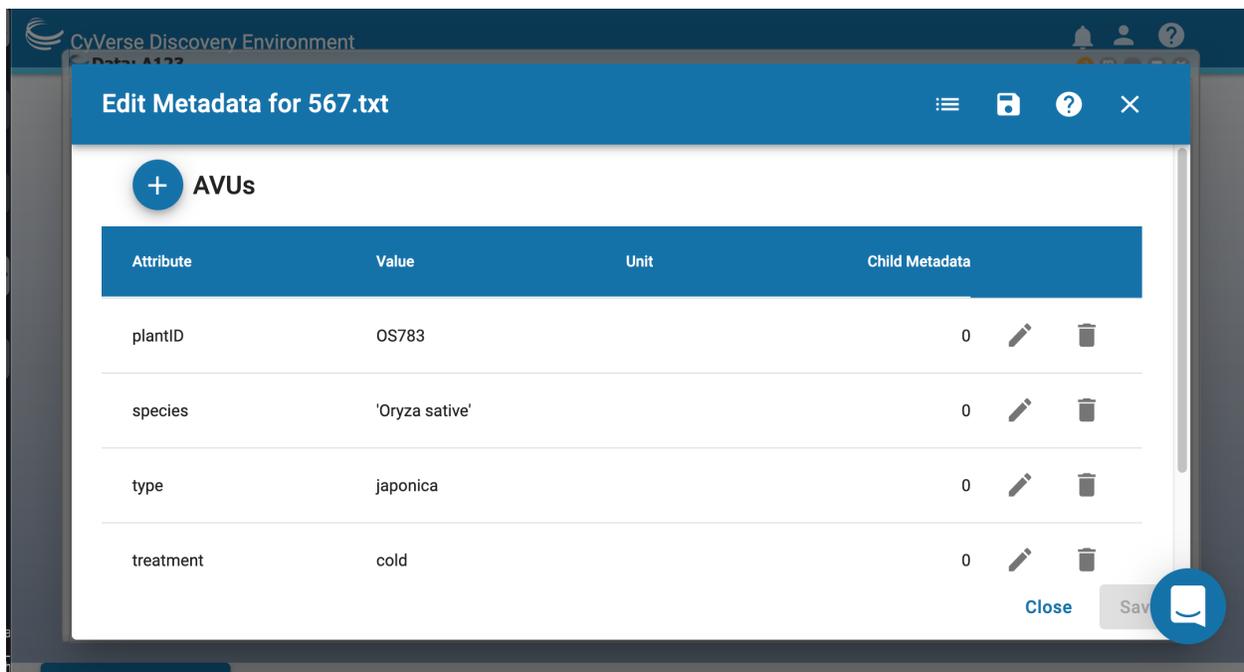
- Create a new text file and share it with someone in the class:



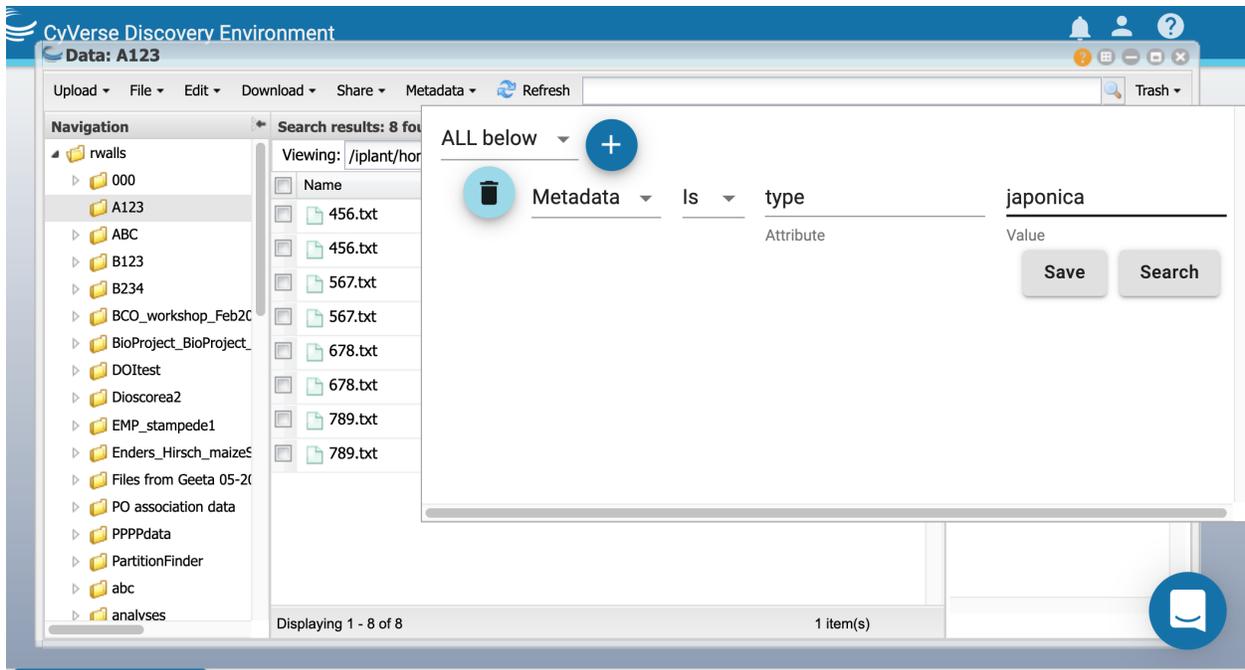
- Download the shared file.

1.23.2 Using metadata in the DE

- Using metadata in the DE :



- Search for data in the DE:



- Try a simple search for the word “maize”
- Try an advanced search for attribute = subject and value = maize
- **Other options to be covered on Tuesday:**
 - bulk metadata application
 - metadata templates

Advanced Metadata Usage

The Data Commons provides advanced metadata features in the Discovery Environment, including:

- metadata templates

Exercise: - Open the DE - Apply a Plant Ontology template to a folder. - Apply an ontology term to a file or folder.

- bulk metadata application

Exercise: 1. Copy the B123 file to your home directory - Using icommands:

```
$ icd /iplant/home/$username
$ icp -r /iplant/home/rwalls/B123 B123
$ ils
```

- In the DE:
 - create a folder in your home directory called **B123**
 - Move into B123
 - Upload >> Import from URL...
 - paste each of the URLs below into a slot. This must be done in two batches.

```
https://raw.githubusercontent.com/CyVerse-learning-materials/foss-2020/
↩master/slides/B123/012.txt
https://raw.githubusercontent.com/CyVerse-learning-materials/foss-2020/
↩master/slides/B123/123.txt
https://raw.githubusercontent.com/CyVerse-learning-materials/foss-2020/
↩master/slides/B123/234.txt
https://raw.githubusercontent.com/CyVerse-learning-materials/foss-2020/
↩master/slides/B123/345.txt
https://raw.githubusercontent.com/CyVerse-learning-materials/foss-2020/
↩master/slides/B123/456.txt
https://raw.githubusercontent.com/CyVerse-learning-materials/foss-2020/
↩master/slides/B123/567.txt
https://raw.githubusercontent.com/CyVerse-learning-materials/foss-2020/
↩master/slides/B123/678.txt
https://raw.githubusercontent.com/CyVerse-learning-materials/foss-2020/
↩master/slides/B123/789.txt
https://raw.githubusercontent.com/CyVerse-learning-materials/foss-2020/
↩master/slides/B123/Rice_metadata.csv
```

2. Apply metadata - In the DE apply the DE apply metadata to the contents of B123

- Browse to B123
- View the metadata for one of the files using the *Metadata* menu or the three dots (it should be blank)
- View the contents of the file *Rice_metadata.csv*
- Browse back to your home directory
- Check the box next to B123
- Select *Metadata > Apply Bulk Metadata*
- Select the file B123/Rice_metadata.csv
- Browse back to B123
- View the metadata of the different files in the directory

3. Advanced search - Click on the search magnifying glass - Click + - Change File Name to Metadata - Under Attribute, type *treatment*, under Value type *cold* - Hit Search

1.23.3 Additional resources

- [DE Guide](#)
- [DE Manual](#)
- [Using CyVerse for a shared project](#)

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



1.24 Discovery Environment - Data Analysis

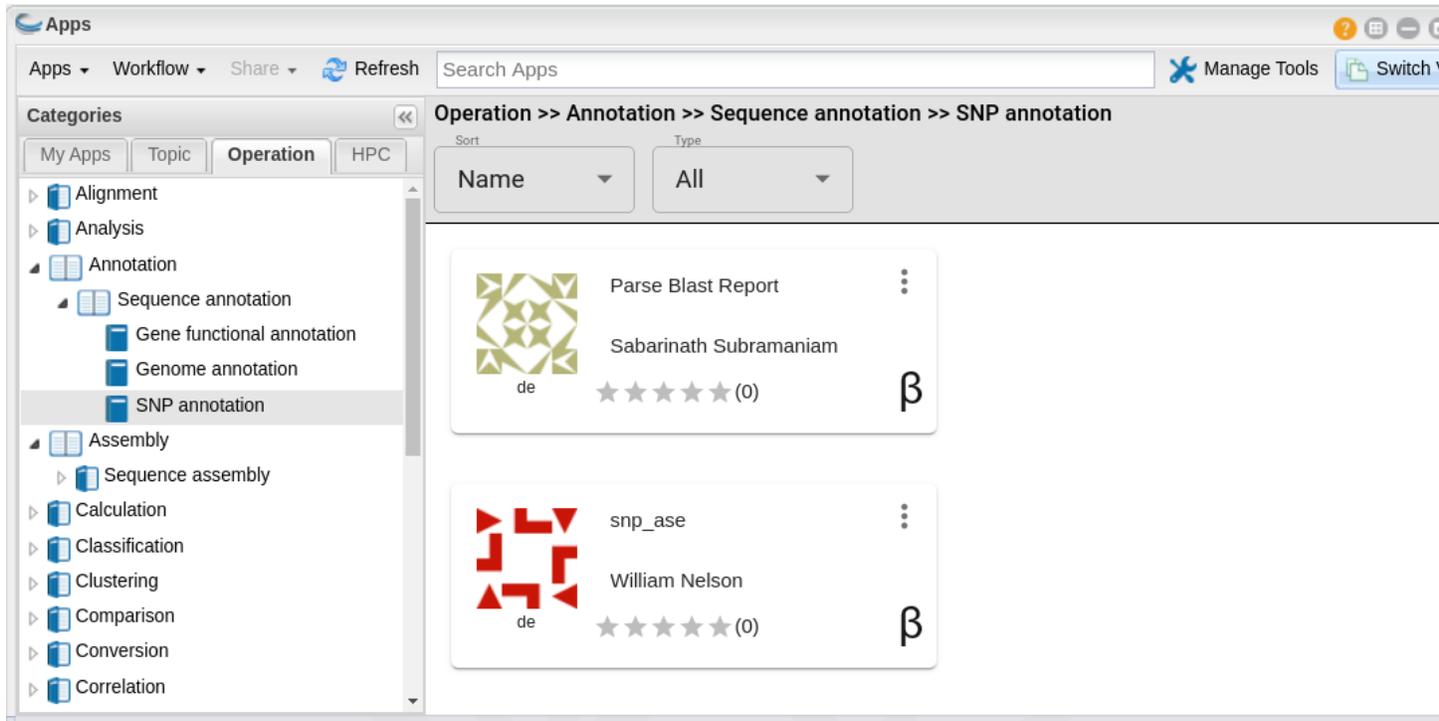


1.24.1 Why use the DE?

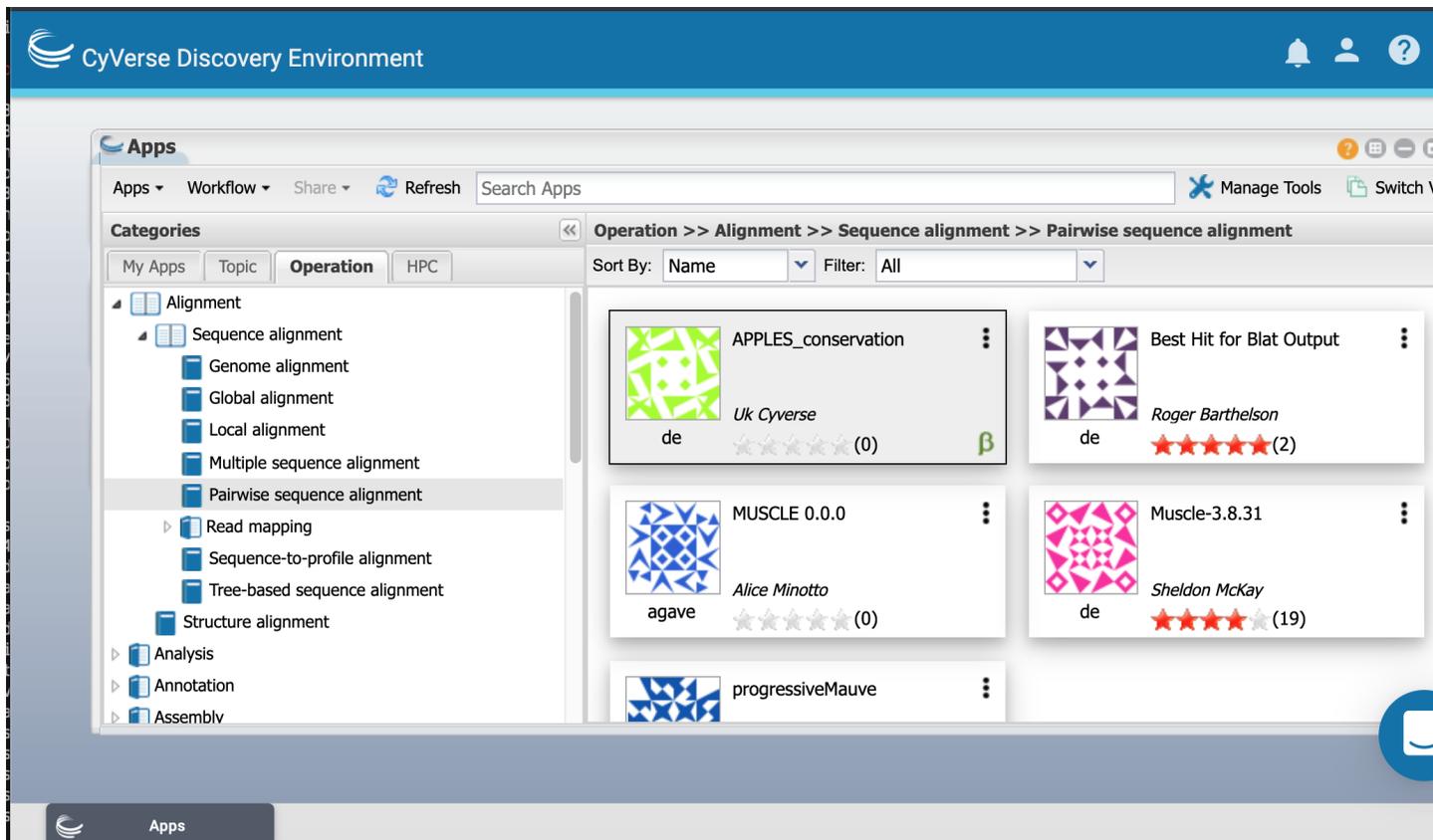
- Use hundreds of bioinformatics Apps without the command line
- Executable and interactive modes
- Seamlessly integrated with data and high performance computing – not dependent on your hardware
- Create and publish Apps and workflows so anyone can use them
- Analysis history and provenance – “avoid forensic bioinformatics”
- Securely and easily manage, share, and publish data

Apps Window

Customizing the Apps window view



1.24.2 Finding Apps



When adding new apps to the DE, developers have the option of adding the app to a community. Users can join a community in the ‘communities’ menu under the person icon in the top right corner of the DE window. A new category will then be added to the left pane of the ‘Apps’ window called ‘My Communities’. This can be an easy way to find apps related to a specific project and to see when new apps have been added to that project.

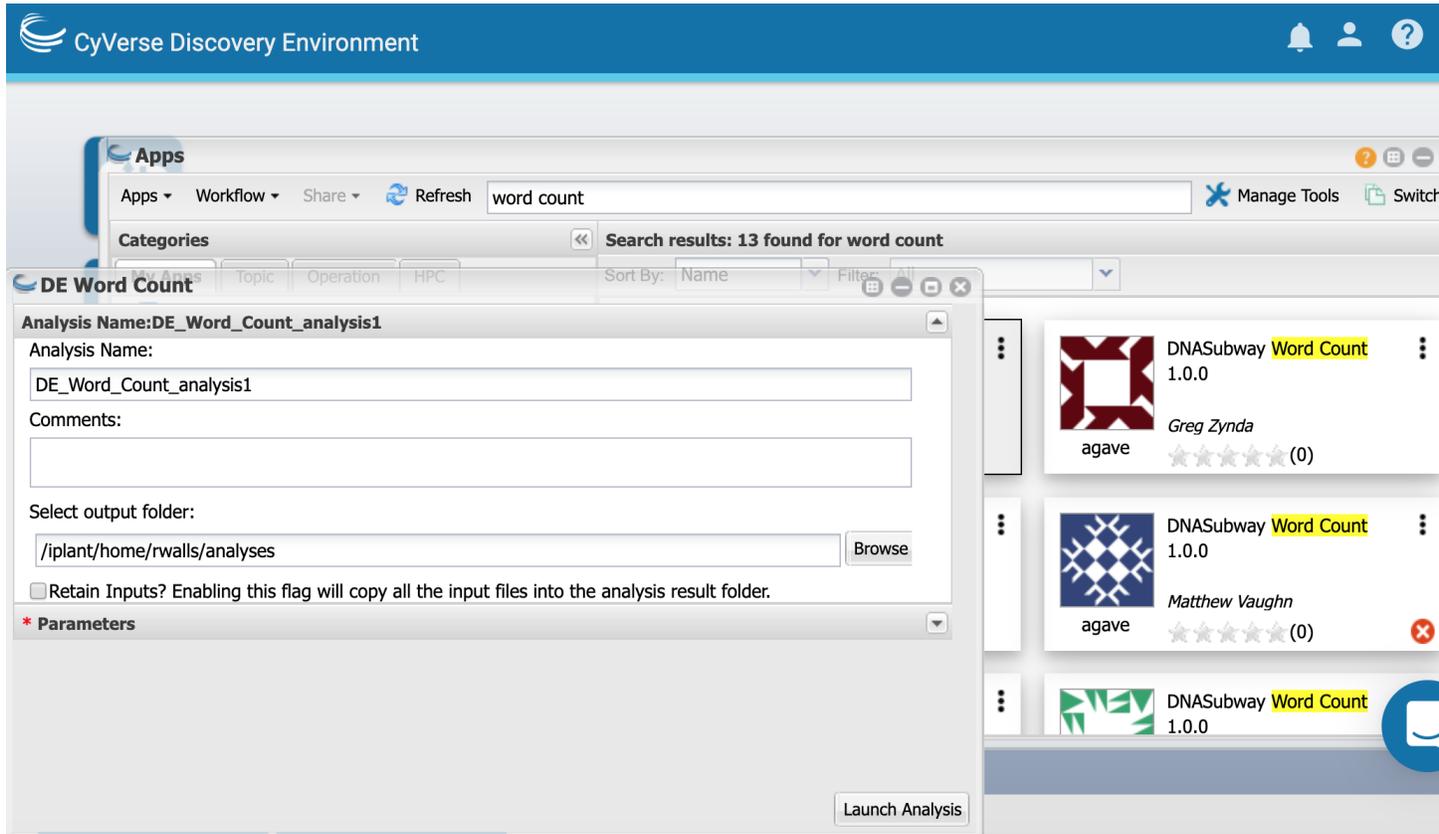
The screenshot shows the 'Apps' window in the CyVerse interface. The left sidebar displays a tree view of categories, with 'My Communities' expanded to show 'i5k'. The main area shows a table of apps under the 'i5k' community. The table has columns for Name, Integrated By, Rating, and System Id. The apps listed are:

<input type="checkbox"/>	Name ↑	Integrated By	Rating	System Id
<input type="checkbox"/>	β combine_GAFs 1.0	Amanda Cooksey	★★★★★ (1)	de
<input type="checkbox"/>	β GOanna 2.0	Amanda Cooksey	★★★★★ (0)	de
<input type="checkbox"/>	β InterProScan Results Fun...	Amanda Cooksey	★★★★★ (0)	de
<input type="checkbox"/>	β KOBAS annotate 3.0.3	Amanda Cooksey	★★★★★ (0)	de
<input type="checkbox"/>	β KOBAS annotate and iden...	Amanda Cooksey	★★★★★ (0)	de

1.24.3 Types of apps

- **Executable:** user starts an analysis and when the analysis finishes they can find the output files in their ‘Analyses’ folder
 - **DE:** run locally on our cluster
 - **HPC:** labeled as ‘Agave’ in the DE. Run on XSEDE resources at Texas Advanced Computing Center (TACC)
 - **OSG:** run on the Open Science Grid
- **Interactive:** also called Visual and Interactive Computing Environment (VICE). Allows users to open Integrated Development Environments (IDEs) including RStudio, Project Jupyter and RShiny and work interactively within them.

Launch an executable analysis



Analyses window

The analysis window provides the status (submitted, running, failed, completed) of each of your analyses.

The screenshot shows the 'Analyses' section of the CyVerse Discovery Environment. At the top, there is a navigation bar with the CyVerse logo and the text 'CyVerse Discovery Environment'. Below this, there is a sub-header 'Analyses' with a search bar and filters for 'View' (set to 'All') and 'App Type' (set to 'All'). The main content is a table with the following columns: Name, Owner, App, Start Date, End Date, and Status. The table lists five analyses:

Name	Owner	App	Start Date	End Date	Status
JupyterLab-0.0.2_VernetData	rwalls	JupyterLab-0.0.2	2018-11-07 14:39:17	2018-11-07 16:56:39	Completed
shiny-0.10.2.2_analysis1	rwalls	shiny-0.10.2.2	2018-11-07 10:48:29	2018-11-07 11:00:33	Canceled
RAML_Start_Tree_8.2.11_49-1	rwalls	RAML Start Tree 8.2.11	2018-08-22 16:47:58	2018-08-22 16:55:23	Completed
Parser_3.0.20_analysis1	rwalls	Parser 3.0.20	2018-08-22 11:26:47	2018-08-22 13:01:39	Completed
SSAdemo	rwalls	JupyterLab	2018-08-13	2018-09-18	Failed

A variety of information and tasks related to an analysis can be found in the 'three dots' menu at the right.

- [Relaunch or cancel an analysis](#)
- [Troubleshoot an analysis](#)
- [Share an analysis](#) with a collaborator.

Launch an interactive analysis (VICE)

Visual Interactive Computing Environment VICE introduces graphic user interfaces (GUIs) and common Integrated Development Environments (IDEs) such as Project Jupyter Notebooks & Lab, RStudio, Shiny Apps and Linux Desktop

Additional resources

- [DE Guide](#)
- [DE Manual](#)
- [VICE Manual](#)
- [Using CyVerse for a shared project](#)

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



1.25 Discovery Environment - Tools & Apps

- **CyVerse tool:** Software program that is integrated into the back end of the DE for use in DE apps
-
- **CyVerse app:** graphic interface of a tool made available for use in the DE

The (containerized) tool must be integrated into the Cyverse DE first. Then an app (interface) can be built for that tool.

1.25.1

1.25.2 Building an App for Your Tool

You can build an app for any tool that:

- is private to you
- is shared with you
- is public

Note: It is a good idea to check to see if the tool you want is already integrated before you start. The tool may be there already and you can build an app using it.

In the 'Manage Tools' window search for 'porechop' in the search bar at the top of the window. Select the porechop public tool and choose 'Use in App' from the 'Tools' menu

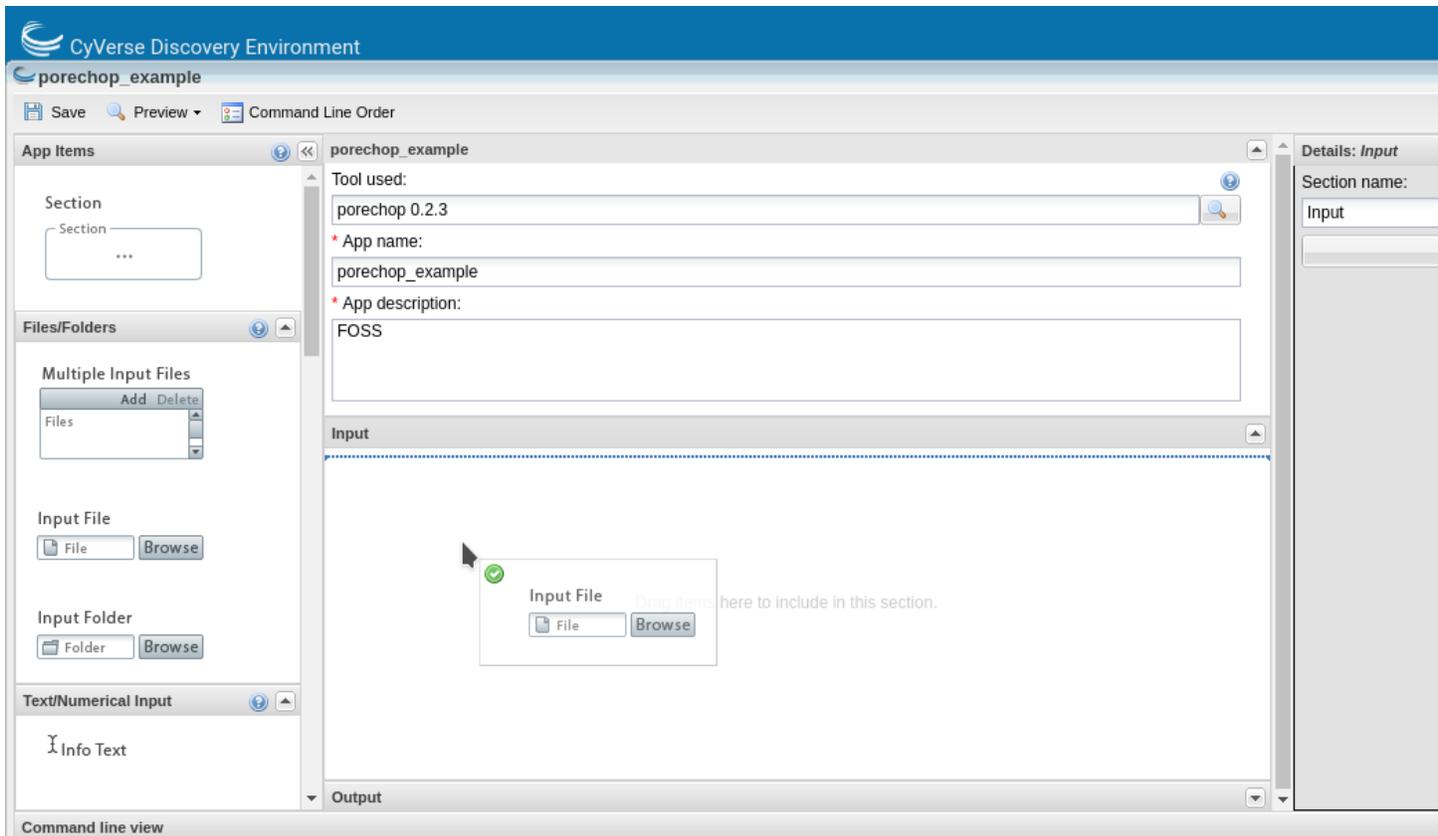
The screenshot displays the 'Manage Tools' interface in the CyVerse Discovery Environment. The main area contains a table of tools with the following data:

Image name	Tag	Status
kapeel/ballgown-r-package		Public
kapeel/helitron		Public
kapeel/sine	latest	Public
bjoyce3/simpriyhtfilesetup	1.0	Public
LncReporter	1.0	Public
Kallisto	latest	Public
Gffcompare	latest	Public
ht-path-list-splitter	latest	Public
portalmehpc_createkeys	docker.cyverse.org/portalmehpc_createkeys 3.0.0	read
Multiqc	pvdb90/multiqc	latest Public
lastdb	docker.cyverse.org/last	latest Public
rmta	docker.cyverse.org/rmta	v1.5 Public

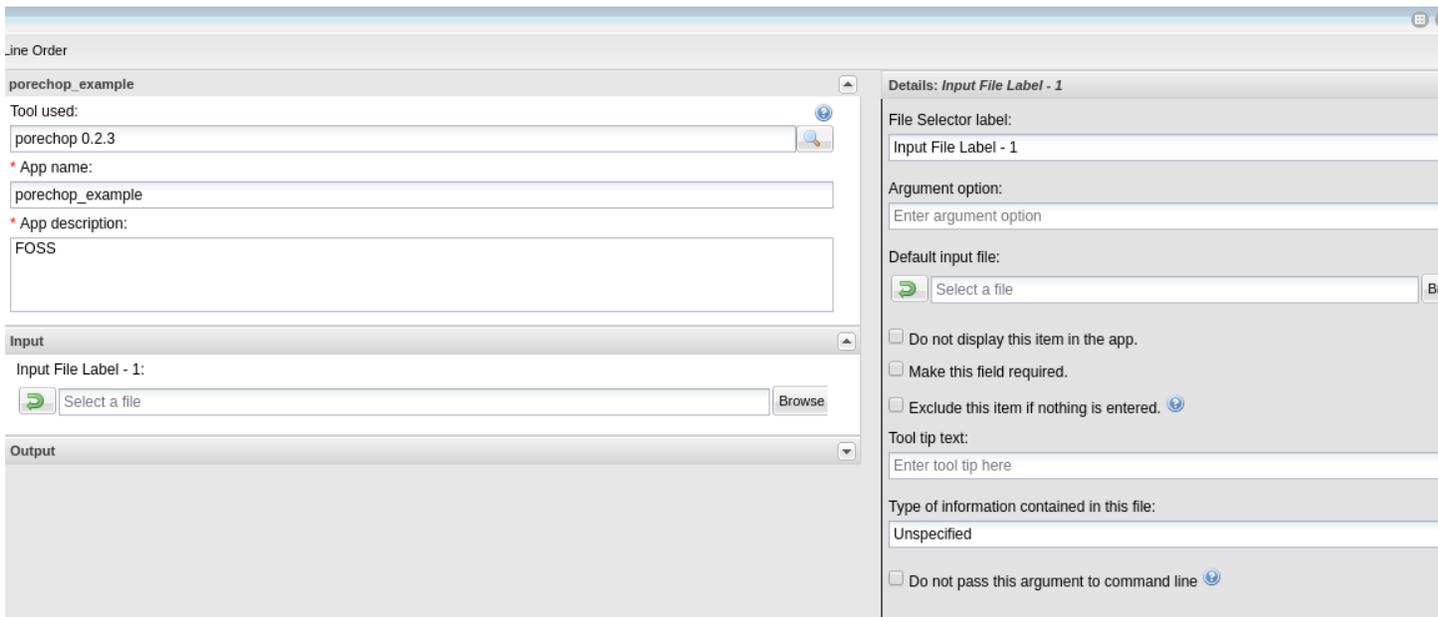
The context menu is open, showing the following options:

- Add tool...
- Request Tool
- Edit
- Delete
- Use in app

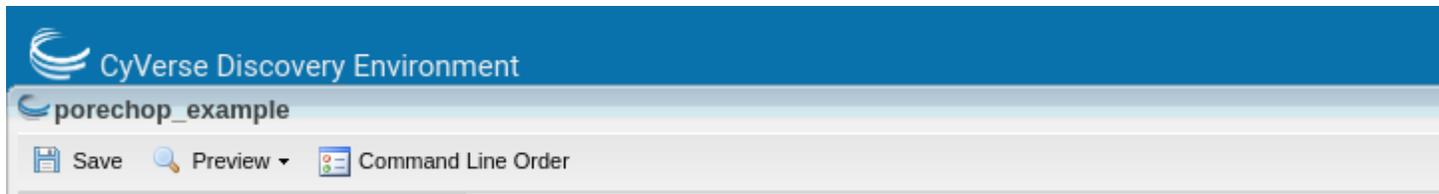
This will open the 'Create App' window. The tool to use will be pre-populated. Choose an informative app name and description (eg. tool name and version). Apps features can be added by dragging the feature from the left pane into the center pane.



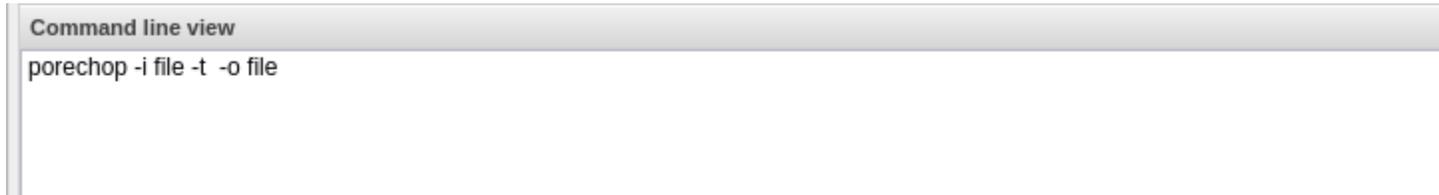
You can edit the details of an app feature by selecting it in the center pane and editing in the right pane. Divide the app into sections appropriate for that tool (input, options and output are usually sufficient sections for simple apps).



For each option you add, you will need to specify what the option is, the flag (if there is one) and whether that option is required. If an option is not required be sure to check the 'exclude if nothing is entered' box. For tools that have positional arguments (no flags, eg. -z) you can modify the order of the commands by clicking the 'command line order' at the top of the window.

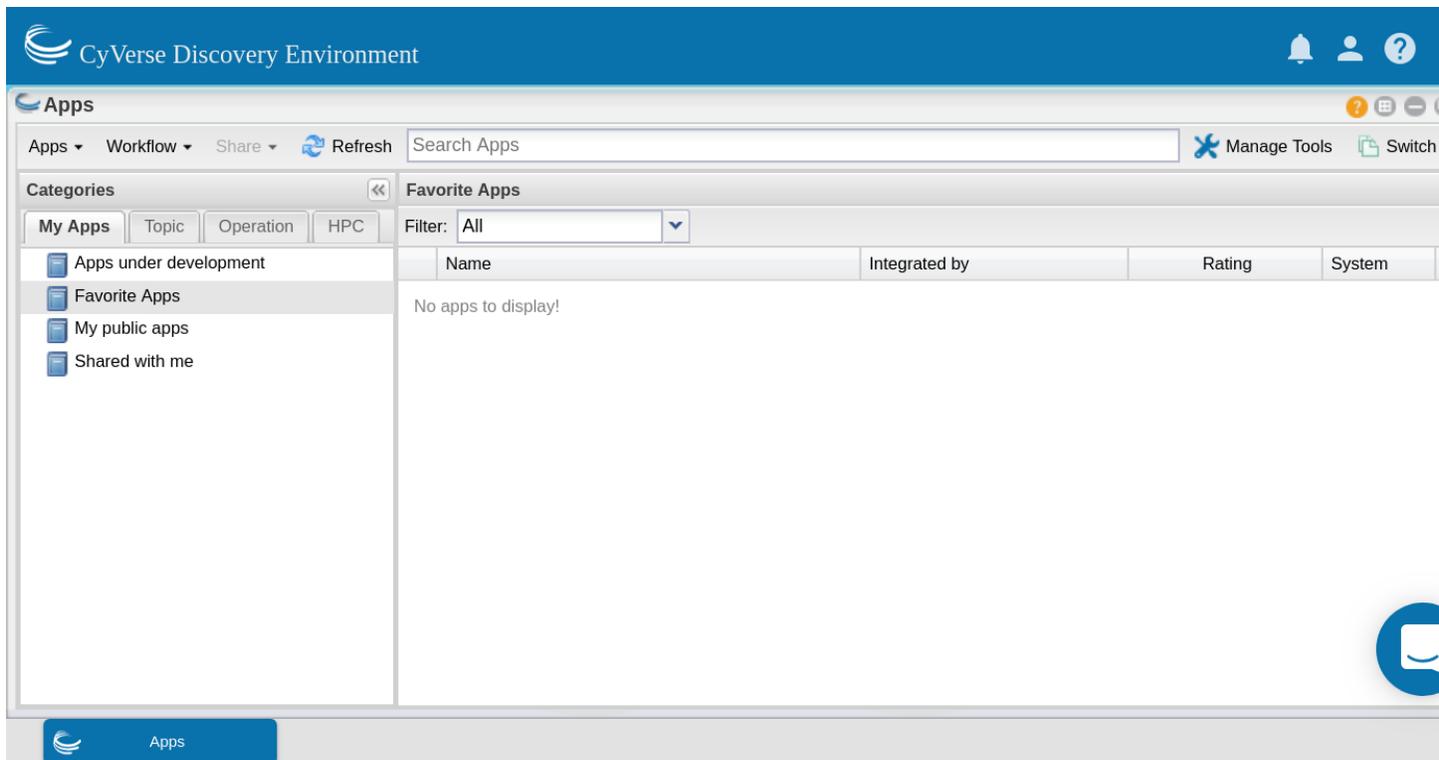


As you add options to your app you will see in the bottom pane (command line view) what the command would look like on the command line.



Although it is best to add all of the options for your tool, as it makes the app the most useful, you can expose as many or as few options as you like (as long as you add all the required options). Once you have finished adding options click save and close your app.

Now test your app with appropriate data. Your app can now be found in the 'My apps in development' category of the 'Apps' window (which displays by default).



Once you know your app works correctly you can share or publish it as you wish. Public apps must have example data located in an appropriately named folder here:

```
/iplant/home/shared/iplantcollaborative/example_data
```

All public apps also have a brief documentation page on the [CyVerse Wiki](#)

To publish your app click on 'Share' at the top of the 'Apps' window and select 'Make public'. You will need to

supply a:

- Topic (eg. genomics)
 - Operation (eg. assembly)
 - location of the example data
 - brief description of inputs, required options and outputs
 - link to CyVerse Wiki documentation page
 - link to documentation for the tool (provided by the developers)
-

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org
 - # Use this example to ensure that links open in new tabs, avoiding # forcing users to leave the document, and making it easy to update links # In a single place in this document



1.26 Discovery Environment - VICE

CyVerse data science workbench, the *Discovery Environment*, includes a feature called *VICE* (*Visual Interactive Computing Environment*)

VICE uses Docker containers to launch interactive programs, like RStudio, R Shiny Apps, Project Jupyter, Data Mining, and WebGL Applications that can be run in a browser. These programs allow users to interact with their data and do analyses in one place (i.e. view outputs in the same window code is executed). Researchers using VICE can explore their datasets interactively in the Discovery Environment while using the Data Store.

While VICE does require some coding knowledge, it is for anyone who wants to interact with data in an iterative way.

Visual Interactive Computing Environment

- You can launch existing VICE images from the DE, or integrate your own using the Manage Tools.
 - VICE apps are containers, and your data are in the container until you move them off of it. Your results will be saved when the app terminates in your `/username/analyses` directory, unless you specify that the app results be saved elsewhere.
-

1.26.1 Create a VICE app

1. Find the Docker image of your interactive tool/software of your interest

We will integrate RStudio as a VICE app in DE. There is a Docker image available for RStudio on Dockerhub.

Note: If there is no Docker image available for your tool of interest, then you would either find a Dockerfile which you can use to build the Docker image for your tool of interest or create one. You can get more help with either of that from [here](#)

2. Test the Docker image locally on your computer (Optional but recommended)

This is optional but highly recommended step to confirm that the Docker image for your tool of interest is working as expected.

2.1 Pull the image from Dockerhub

```
$ docker pull cyversevice/rstudio-verse:3.6.0
```

You could also use [Play with Docker](#), which is a free resource to test and run the Docker container.

2.2 Sample run

```
$ sudo docker run --rm -v /$HOME:/app --workdir /app -p 8787:80 -e REDIRECT_URL=YOUR_
↪IP_ADDRESS:8787 cyversevice/rstudio-verse:3.6.0
```

In your browser address bar, type `YOUR_IP_ADDRESS:8787` to access RStudio

Note: The username and password for Rstudio is `rstudio` and `rstudio1` respectively

Once you can open the Rstudio, then you are ready to integrate the tool in DE

3. Add tool in CyVerse Discovery Environment

3.1 Log-in to CyVerse Discovery Environment and click on the “Apps” window

3.2 Click “Manage Tools” -> “Tools” -> “Add Tool” and fill the details for your Docker image

Fill out the following details

```
Tool-Name: FOSS-Rstudio
Description: RStudio VICE app
Version: 3.6.0
Type: interactive
Image Name: cyversevice/rstudio-verse
Docker Hub URL: https://hub.docker.com/repository/docker/cyversevice/rstudio-verse
Tag: 3.6.0
Working Directory: /home/rstudio
Port Number: 80
```

Container Image ^

Image Name*

If the image is in Docker Hub, this field should be in username/image-name format, where username is your Docker Hub username. If it's in another registry, such as the CyVerse registry, it should be in registry-host/image-name format.

Docker Hub URL

Tag*

Container Name

WARNING: Do not add a tool without an Entry Point setting if its Docker image also does not have a default `ENTRYPOINT`. If a tool like this is required, then its Network Mode setting should be set to `none` to contain any risky scripts run by this tool.

Entrypoint

Working Directory

UID

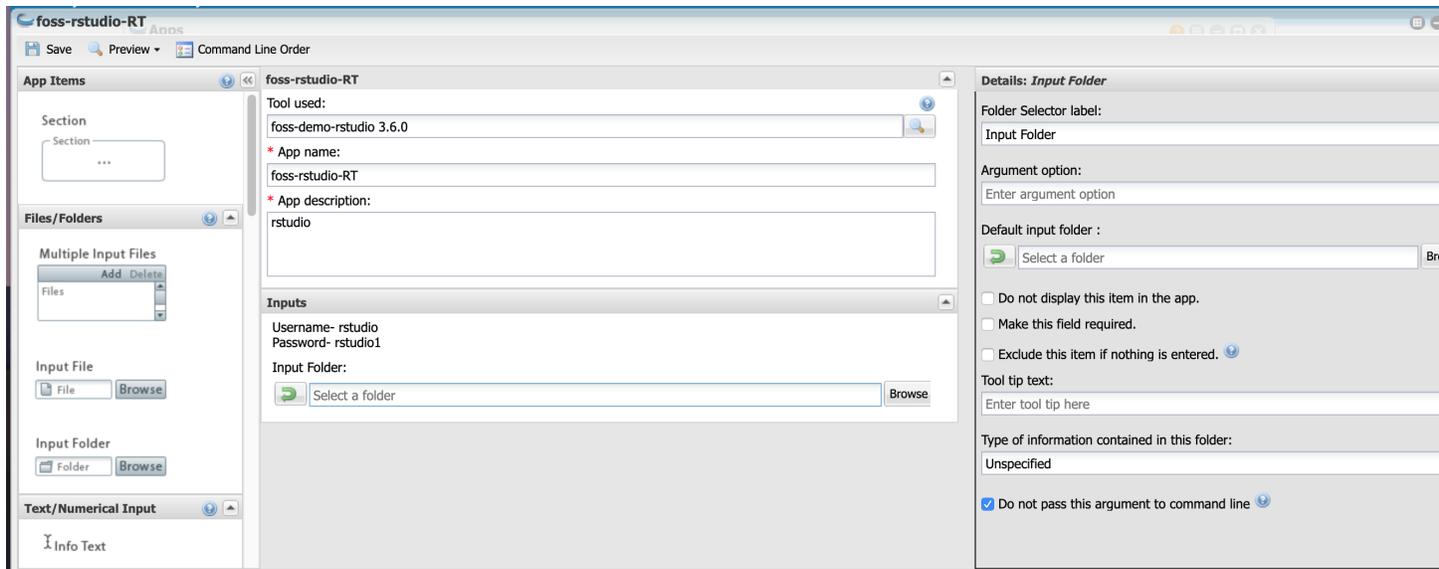
Container Ports ^

+ Port

Container Port	Host Port	Bind to Host
<input type="text" value="80"/>	<input type="text" value="Port Number"/>	<input type="checkbox"/> Bind to Host

3.3 Create a VICE app

Click on the “Apps” window. Click “Apps” -> “Create New”



Note: For VICE apps, make sure to check “Do not pass this argument to command line”

Example data: /iplant/home/shared/iplantcollaborative/example_data/FOSS_2020/vice_exampdata

1.26.2 Version control using Git within RStudio

Step1: Make a new repo on GitHub

- Create a New repository on GitHub. Click the green “New” button to create a new repo. Enter the name of the repo and check initialize this repo with a README file. Click “Create Repository” button.

Repositories



Find a repository...

Step2: Clone the new GitHub repo

- From Github, copy the repo URL via the green “Clone or Download” button.
- In RStudio, start a new project. File > New Project > Version Control > Git. Paste the GitHub URL in the “repository URL”.
- Enter the name of the “project directory” which should be same as the name of your GitHub repo.
- Click “Create Project”.

Step3: Push your changes to GitHub

- Make some changes (to README file or add a new code)
- Click on Git (Check Git tab on upper right corner of your RStudio window)
- Check the files that you want to commit. Click commit and enter a commit message on the next window.

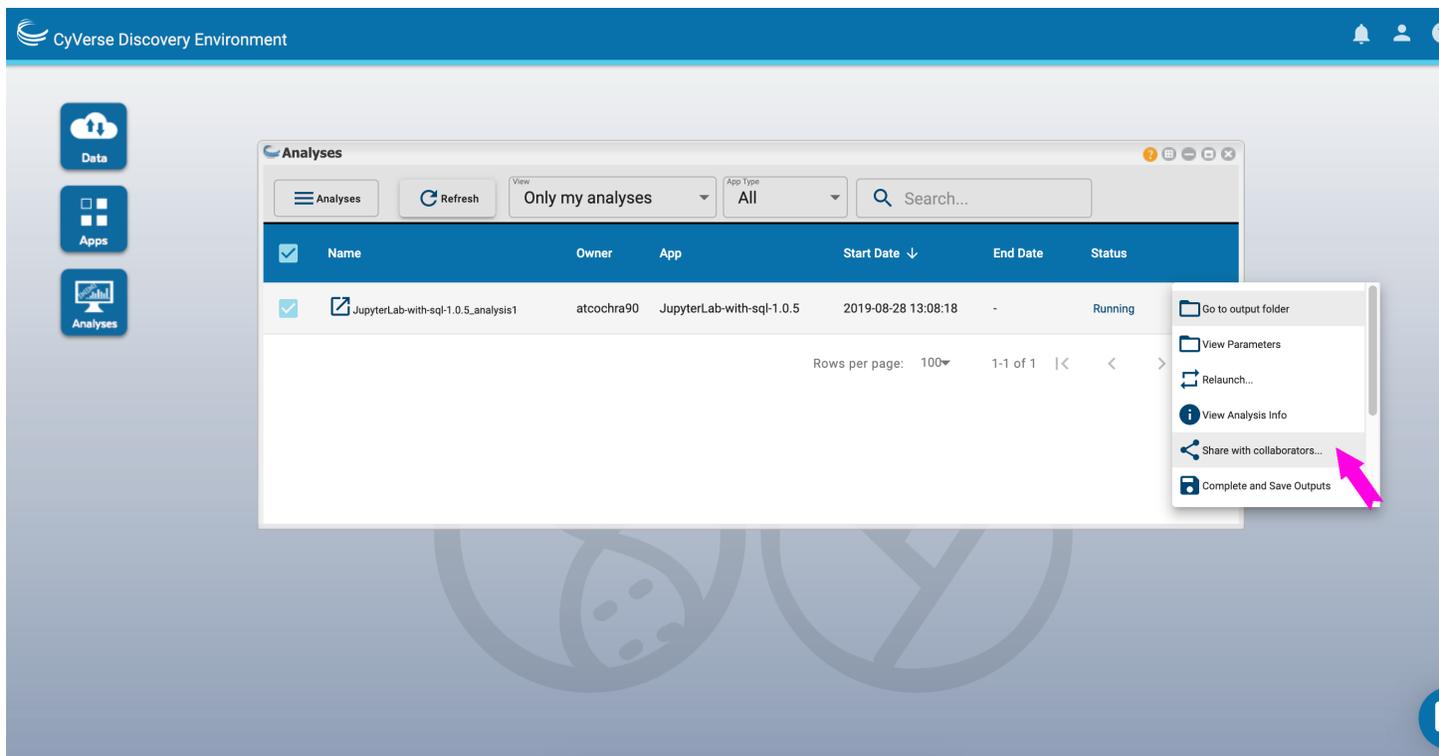
- After you commit, it will prompt you to set your account's default identity. Go to terminal and configure your git user name and email using the following commands. You should be able to commit without any problems after you configure.
 - `git config --global user.email "you@example.com"`
 - `git config --global user.name "Your Name"`
- To push the changes to the remote GitHub repository press the Push button on the upper right corner of the commit window. You will be prompted to enter the username and password of your GitHub account.



1.26.3 Sharing VICE apps with collaborators

You can share your VICE workspace with colleagues (with a CyVerse account) who can see and edit your notebooks, logs, and outputs.

- To share your workspace

A screenshot of the CyVerse Discovery Environment interface. The top navigation bar is blue with the CyVerse logo and the text 'CyVerse Discovery Environment'. On the left, there are three icons: Data, Apps, and Analyses. The main content area shows a table titled 'Analyses'. The table has columns for Name, Owner, App, Start Date, End Date, and Status. There is one row with the following data: Name: JupyterLab-with-sql-1.0.5_analysis1, Owner: atcochra90, App: JupyterLab-with-sql-1.0.5, Start Date: 2019-08-28 13:08:18, End Date: -, Status: Running. A context menu is open over the row, showing options: Go to output folder, View Parameters, Relaunch..., View Analysis Info, Share with collaborators... (highlighted with a pink arrow), and Complete and Save Outputs.

- Opening workspaces shared with you

The screenshot displays the CyVerse Discovery Environment interface. On the left, there are navigation icons for Data, Apps, and Analyses. The main area shows a window titled 'Analyses' with a table of analysis jobs. A pink arrow points to the first row of the table.

<input type="checkbox"/>	Name	Owner	App	Start Date ↓	End Date	Status	
<input checked="" type="checkbox"/>	JupyterLab-with-sql-1.0.5_analysis1_test	upendra_35	JupyterLab-with-sql-1.0.5	2019-08-27 13:07:40	-	Running	⋮

Below the table, it shows 'Rows per page: 100' and '1-1 of 1' with navigation arrows.

The browser address bar shows the URL: `a4ef97137.cyverse.run/lab?`. The JupyterLab interface is open to a notebook titled 'SQL_tutorial.ipynb'. The notebook content includes:

SQL Interface to JupyterLab

1. Using IPython SQL Magic extension

Magic commands are a set of convenient functions in Jupyter Notebooks that are designed to solve some of the common problems in standard data analysis. You can see all of the available magics with the help of `%lsmagic`.

IPython SQL magic extension makes it possible to write SQL queries directly into code cells as well as read the results straight into pandas DataFrames (Source). This works for both the traditional notebooks as well as the modern Jupyter Labs.

```
[1]: # Loading the SQL module
     %load_ext sql
```

The above magic command loads the ipython-sql extension. We can connect to any database which is supported by SQLAlchemy. Here we will connect to a SQLite database. Enter the following command in the code cell

```
[2]: %sql sqlite://
```

```
[2]: 'Connected: @None'
```

If you get the output as `Connected: @None`, this means the connection has been established.

- **Creating a database**

Finally, we create a demo table called `EMPLOYEEES` to showcase the function.

```
[3]: %%sql
     CREATE TABLE EMPLOYEE(firstname, lastname);
     INSERT INTO EMPLOYEE VALUES ("Tom", "Mitchell");
```

1.26.4 Specific instructions for launching VICE applications

-Jupyter lab

-Rstudio

-Rshiny

List of Discovery Environment VICE apps

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



 [Learning Center Home](#)

1.27 Introduction to Cloud Computing

The simplest way to think of cloud is that its like a laptop or desktop computer that you connect to remotely. In more granular applications, cloud is a way of submitting a single task to a computer or a set of computers on demand without having to host them or keep them running. Cloud services can also be optimized to run many machines in parallel for large cluster-like tasks, similar to what is done using High Performance Computing.

There are three major private cloud providers: [Amazon Web Services \(AWS\)](#), [Google Cloud Platform \(GCP\)](#), and [Microsoft Azure](#). These services cost money to use. They do provide free credits to researchers ([GCP](#), [AWS](#), [Azure](#)) with short applications.

The big cloud providers have been replicating publicly owned data sets, e.g. [40+ years of NASA and European Space Agency \(ESA\) earth observation system data](#), on their cloud-hosted data storage services in the hope that researchers and businesses will pay to use these data by doing cloud computing on them. [NASA recently announced a plan to move hundreds of PBs of its data to AWS](#).

Some cloud services are free, like [Google's Earth Engine](#), others have limited sandboxes which are useful for training, but may not fit your needs for larger scale data analyses. Launching IDEs like Jupyter Notebooks or RStudio in cloud is possible with platforms like [MyBinder](#) and [CoLab](#).

There are also options where your institution can stand up a cloud service on its own hardware: [Open Stack](#) and [VMWare](#). CyVerse and XSEDE operate multiple OpenStack clouds which they provide as a service called [Atmosphere \(CyVerse\)](#) and [Jetstream \(XSEDE\)](#) free to researchers.

We will be focusing the workshop time on the applications of 'containers' like [Docker](#) and [Singularity](#) which are a way of taking your research and running analyses on ANY cloud provider. The rapid development of containers and

container orchestration is due to the rise of the cloud. The utility of containers to researchers in the areas of sharing and reproducible research are fortunate benefits.

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



1.28 Atmosphere

CyVerse operates a cloud service called *Atmosphere*. Users can request up to 5,000 allocation units [units are hours (hr)] per month. E.g. a 1-core instance uses 1 AU/hr, a 4-core instance uses 4 AU/hr, and a 16-core instance uses 16 AU/hr. Allocations are automatically reset to 128 AU on the 1st of each month.

Some things to remember about the platform

- Allocation Units can be requested each month, but expire at the end of the month. Users can request more AU by clicking the Request More Resources button in the Atmosphere UI. You can also get help by asking questions in the Intercom (blue button in the lower right of the CyVerse website pages).
- Don't leave VMs running that aren't being used (be a good data science neighbor and free resources for others).

1.28.1 Virtual Machines on Atmosphere or Jetstream

Provision VM

The screenshot shows the CyVerse Provision VM interface. At the top, there is a blue navigation bar with the CyVerse logo and menu items: Dashboard, Projects (selected), Images, and Help. The user 'tyson_swetnam' is logged in. Below the navigation bar, there are three tabs: RESOURCES (selected), DETAILS, and OPTIONS. The main content area displays the project name 'NEON Data Institute 2018' and three buttons: 'NEW', a refresh icon, and a 'Request more resources' button (indicated by a tooltip). Under the 'Instances' section, there is a table with one instance:

<input type="checkbox"/>	Name	Status	Activity	IP Address	Size	Provider
<input type="checkbox"/>	 Ubuntu 16_04 GUI XFCE Base	● Active	N/A	128.196.142.9	Medium3	CyVerse Cloud - Marana

Below the instances table, there is a 'Volumes' section with a table header:

<input type="checkbox"/>	Name	Status	Size	Provider
--------------------------	------	--------	------	----------

Request Resources

What resources would you like to request?

E.g 4 CPUs and 8GB memory, running 4 cores for 1 week, an additional 500 AU, etc.

How will you use the additional resources?

E.g. To run a program or analysis, store larger output, etc.

CANCEL REQUEST RESOURCES

Login

Log into [CyVerse Atmosphere](#)

[Atmosphere Manual](#)

Alternately, log into [XSEDE Jetstream](#)

Fill in your username and password and click “LOGIN”

Create a Project

This is something you only need to do once.

- Click on the “Projects” tab on the top and then click “CREATE NEW PROJECT”
- Enter a name, e.g. “FOSS2019” into the Project Name field.
- the Description can be something complex and long (like an extended abstract, or tutorial), or something short like “CyVerse FOSS 2019”.

- Select the newly created project

Start a new Instance

From your Project folder, you can select “New” and “Instance”

1. Suggest you select a featured image with a Graphic User Interface (GUI).

Suggested Atmosphere Image(s):

Atmosphere Image(s):

Here are the tested Ubuntu images.

Warning: The latest version of Ubuntu (18.04) may not have current packages for some software.

Image Name	Version	Description	Link
Ubuntu 18.04 GUI	1.0	Ubuntu 18.04 GUI XFCE Base	Image
Ubuntu 18.04 non-GUI	1.0	Ubuntu 18.04 non-GUI Base	Image

- Find the “Ubuntu 18.04” image, click on it
- Give it a short name that is distinct “my_first_vm”
- Select ‘tiny1 (CPU: 1, Mem: 4GB, Disk: 30GB)’. Because this is your first attempt at provisioning a virtual machine it doesn’t need to be a workhorse (yet).
- Leave rest of the fields as default.
- Wait for it to become active
- Be Patient (but not too patient - if it takes >10 minutes the system may be at capacity, if you’re trying to launch a large or extra large VM, try something smaller).
- You can click on your new instance to get more information.

Accessing the Shell

Once the instance is *active*, you can access it via `ssh` or by using the Web Shell provided by Atmosphere.

- Click “Open Web Shell”, *or*, if you know how to use `ssh`,

you can `ssh` in with your CyVerse username on the IP address of the machine

```
ssh CyVerseUserName@<INSTANCE-IP-ADDRESS>
```

You should see something like this

```
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.4.0-81-generic x86_64)

Get cloud support with Ubuntu Advantage Cloud Guest:
  http://www.ubuntu.com/business/services/cloud

155 packages can be updated.
0 updates are security updates.

*** System restart required ***
Welcome to
  _ _ _
```

(continues on next page)

- Only create images from the smallest possible versions of your instance. A larger imaged instance cannot be run on a smaller instance.

To request that your instance be imaged click the “Image” button from Actions.

Note: It is advisable to delete the machine if you are not planning to use it in future to save valuable resources. However if you want to use it in future, you can suspend it.

If you want to keep the instance for a future project, you can also “shelve” the instance. It will take a longer period of time to resume a shelved instance.

EZ Installation of Project Jupyter

We install Project Jupyter (Notebooks and Lab) using the [Anaconda distribution](#). Within the Anaconda distribution is the `conda` package manager which can be used to both build and install software.

Anaconda is different than a basic Python installation. It serves as both a package manager and an environment. While this has many benefits, it also adds some complexity to running your Python environments. Still confused? Read about the [myths and misconceptions of Anaconda](#).

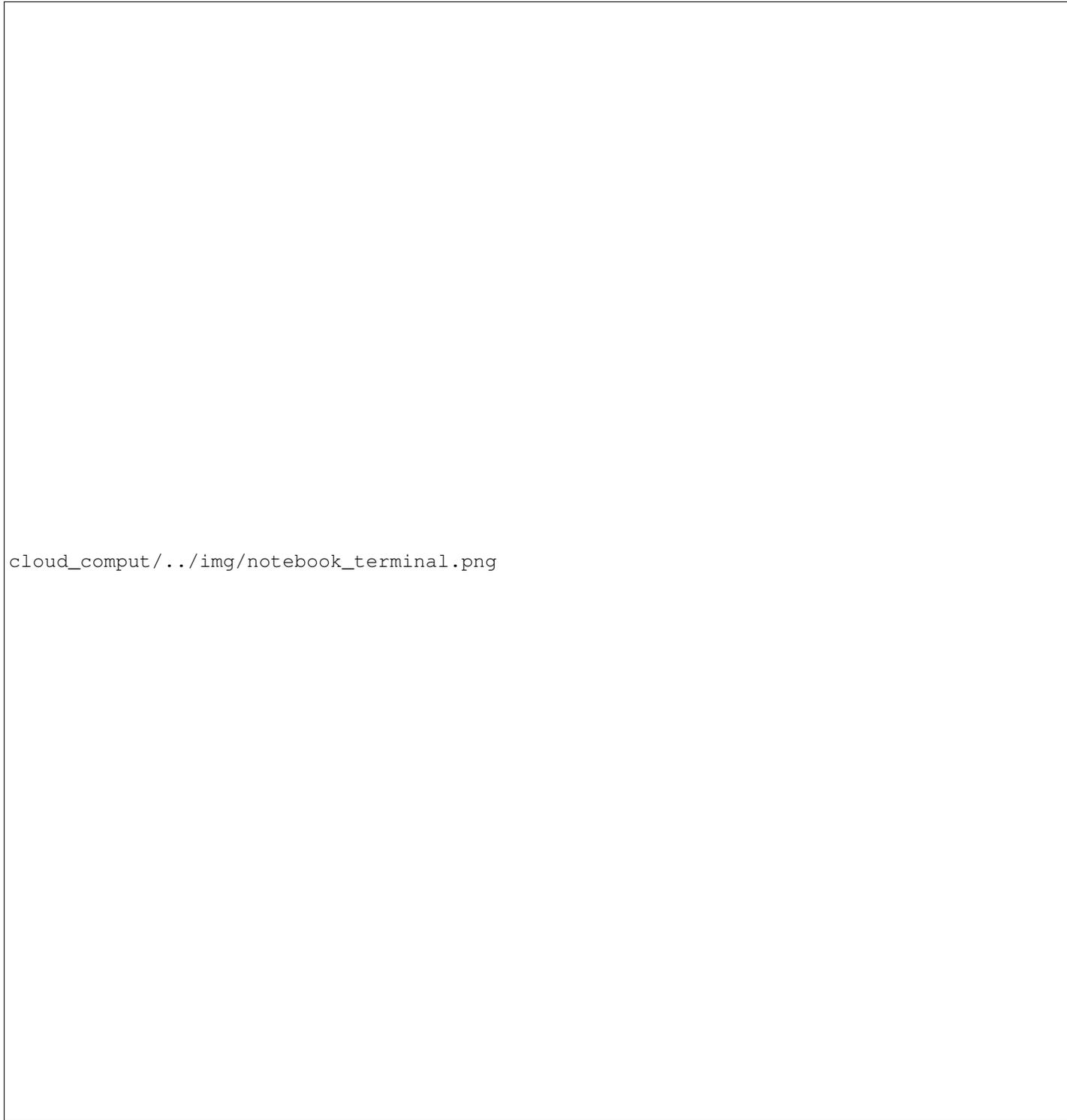
For more details about installing software on Atmosphere visit the [CyVerse Data Science Quickstart Tutorial](#) or the [Jetstream EZ Tutorial](#). There are instructions for `ez` installation of Docker, Singularity, and Anaconda.

If you’re on an instance which already has Anaconda installed, you’ll still need to re-run `ez` to restart the Anaconda virtual environment.

1. Install Anaconda with Python3 (`ez` comes preloaded on featured instances on Atmosphere and Jetstream) by typing:

```
ez j
```

2. Once the installation completes, you’re done! A Jupyter Notebook should now be running on the VM.



cloud_comput/../../img/notebook_terminal.png

3. Click the link showing the notebook URL (notice this is not the localhost:8888).

Note: To install your own packages you'll need to change ownership of the Anaconda installation:

```
sudo chown $(id -u):$(id -g) /opt/anaconda3 -R
```

Down version Python 3.6 to 3.5

To use GDAL you may need to reverse version Python to an earlier version

Kernel installation instructions

```
python -m pip install ipykernel
```

```
conda create -n ipykernel_py35 python=3.5 ipykernel
source activate ipykernel_py35 # On Windows, remove the word 'source'
python -m ipykernel install --user
```

List of Jupyter Kernels

R

```
conda install -c r irkernel
```

JavaScript

```
sudo apt-get install nodejs-legacy npm ipython ipython-notebook
sudo npm install -g ijavascript
ijsinstall
```

Ruby

Add Jupyter PPA

```
sudo add-apt-repository ppa:chronitis/jupyter -y
sudo apt-get update
sudo apt-get install -y iruby
```

Python2 Kernel

```
conda create -n ipykernel_py2 python=2 ipykernel
source activate ipykernel_py2
python -m ipykernel install --user
source deactivate ipykernel_py2
conda activate base # switch back to base Python3 environment
```

Julia Kernel

First, install Julia, here we are installing v0.6.

Once Julia as been installed, run julia from the prompt.

```
wget https://julialang-s3.julialang.org/bin/linux/x64/0.6/julia-0.6.3-linux-
↪x86_64.tar.gz
tar xvzf julia-0.6.3-linux-x86_64.tar.gz
sudo mv julia-d55cadc350/ /opt/julia
rm -rf julia-0.6.3-linux-x86_64.tar.gz
sudo ln -s /opt/julia/bin/julia /usr/local/bin/julia
julia
```

Now, from Julia prompt install the iJulia Kernel.

```
Pkg.add("IJulia")
ENV["JUPYTER"] = "/opt/anaconda3/bin/jupyter"
Pkg.add("Feather")
```

(continues on next page)

(continued from previous page)

```
Pkg.add("DataFrames")
Pkg.add("NamedArrays")
```

Bash Kernel

```
pip install bash_kernel
python -m bash_kernel.install
```

Geospatial dependencies

```
conda install -c conda-forge gdal
```

```
sudo add-apt-repository -y ppa:ubuntugis/ubuntugis-unstable
sudo apt update
sudo apt install gdal-bin python-gdal python3-gdal libgdal-dev
```

Script of Scripts**Official documentation**

```
pip install sos
pip install sos-notebook
python -m sos_notebook.install
```

Installing RStudio-Server

RStudio can be installed in several ways.

First, you can follow the RStudio-Server [instructions for Linux](#)

Second, you can use Docker (following the same [ez documentation](#) as for Anaconda). We suggest using containers from Docker Hub [Rocker](#) on the instance.

```
ezd
sudo usermod -aG docker $USER
exit
docker pull rocker/geospatial
docker run -d -p 8787:8787 rocker/geospatial
```

Third, you can use [Anaconda](#)

Here we use `ezj -R` to install both Anaconda (Jupyter) and R

```
ezj -R
```

This will trigger the Ansible playbook to install `r-base`, `r-essentials`, and a few other commonly used R Data Science packages.

After `ezj -R` has finished, you can install RStudio-Server

Install these misc. dependencies

```
export PATH="/opt/anaconda3/bin":$PATH
sudo chown $(id -u):$(id -g) /opt/anaconda3/ -R
conda update conda
```

(continues on next page)

(continued from previous page)

```
conda install gxx_linux-64
conda install gcc_linux-64
```

Set Path and install gdebi

```
sudo apt-get install gdebi-core
```

Install RStudio-Server with gdebi:

```
echo "export RSTUDIO_WHICH_R='/opt/anaconda3/bin/R'" >> ~/.bash_profile
wget https://download2.rstudio.org/rstudio-server-1.1.447-amd64.deb
sudo gdebi --non-interactive rstudio-server-1.1.447-amd64.deb
```

The installation of RStudio-Server is going to fail because we haven't told it which R to use. Because we are using Anaconda's installation of R, and not the basic installation of R, we have to reassign RStudio to look for Anaconda

```
sudo sh -c 'echo "rsession-which-r=/opt/anaconda3/bin/R" >> /etc/rstudio/
↳rserver.conf'
# export RSTUDIO_WHICH_R='/opt/anaconda/lib/R/bin/R'
# sudo sh -c 'echo "launchctl setenv RSTUDIO_WHICH_R $RSTUDIO_WHICH_R" >> ~/.
↳bash_profile'
```

Restart the server

```
sudo rstudio-server start
```

4. You can launch Jupyter Lab by exiting the notebook and typing `jupyter lab` - but this will allow Lab to only be available on the localhost, with no way to connect from a remote terminal. Exit the notebook by pressing `ctrl + c` twice, and then start a Jupyter Lab.

1.28.2 Hands On

Sateesh's Atmosphere Exercises

Note: To ensure your session doesn't die when you close your terminal use `tmux` or `screen` to start your remote sessions and to detach the screen before exiting.

- detach screen: `ctrl + b` then `d`
 - list tmux sessions: `tmux ls`
 - re-attach screen: `tmux attach -t <session id #>`
-

Establishing a Secure Connection

1. On the VM start the Lab in terminal (don't forget to use `tmux`)

```
jupyter lab --no-browser --ip=* --port=8888
```

Option 1: SSH tunnel

You must have the ability to use `ssh` on your localhost to use this method.

1. Start Jupyter

```
jupyter lab --no-browser --ip=127.0.0.1 --port=8888
```

2. Open a new terminal on your localhost.

```
ssh -nNT -L 8888:localhost:8888 CyVerseUserName@<IPADDRESS>
```

Enter your password when prompted.

The terminal should stop responding after this.

3. In your browser, open a new tab and go to `http://localhost:8888`

Option 2: Caddy

You can use this method with `tmux` in the Web Shell

1. Follow the same step #1 above
2. In the terminal start a new `tmux` session. Then copy/paste the following:

```
echo "$(hostname)
proxy / 127.0.0.1:8888 {
    websocket
    transparent
}
" > Caddyfile
curl https://getcaddy.com | bash -s personal http.nobots
caddy
```

The `Caddyserver` will output a secure URL `https://` for the Atmosphere VM which you can then connect in a new browser tab.

3. Copy / Paste the URL `https://vm142-xx.cyverse.org` into a new browser tab.

Description of output and results

Congratulations - you've got a Virtual Machine ready to do some serious data science!



Learning Center Home

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



CYVERSE®



Learning Center Home

1.29 Research Cyberinfrastructure associated with CyVerse

While CyVerse provides free and open resources for data storage and computing, you may find that your work is pushing the limits of what CyVerse offers in its core services.

By design, there are other national research organizations whom you should be aware of, and ready to interface with. CyVerse is a partner with numerous other organizations, and can be used as a nexus to send your data and algorithms to the largest public research computing centers in the world.

1.29.1 XSEDE

The [eXtreme Science and Engineering Discovery Environment \(XSEDE\)](#) is a single virtual system that scientists can use to interactively share computing resources, data and expertise. People around the world use these resources and services — things like supercomputers, collections of data and new tools.

Exercise

1. Sign up for an XSEDE account via the [User Portal](#)
2. Determine whether you're ready for a start up allocation on one of the XSEDE resource.
3. Fill out an XSEDE startup allocation.

Jetstream

Jetstream is CyVerse Atmosphere deployed on XSEDE. Allocations can be requested using the XSEDE portal.

With Jetstream, you can launch larger VMs than on CyVerse (currently up to 44-cores) with an option for GPU computing in the near future.

[Jetstream API Documentation](#)

TACC

CyVerse resources are mirrored between the University of Arizona and the [Texas Advanced Computing Center\(TACC\)](#). You can access TACC resources from CyVerse, or log directly into their infrastructure via XSEDE.

Open Science Grid

The [OpenScienceGrid \(OSG\)](#) is another CyVerse partner. You can use the OSG through our Discovery Environment to launch high throughput computing jobs.

1.29.2 CyVerse Powered By

CyVerse supports “Powered by” projects which utilize more of the available CyVerse cyberinfrastructure than are exposed through its public services.

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



1.30 Other Cyberinfrastructure Projects

1.30.1 National Groups

The [Cloud Native Computing Foundation](#) is an open source software foundation dedicated to making cloud native computing universal and sustainable

The [Open Science Framework](#) is a tool that promotes open, centralized workflows by enabling capture of different aspects and products of the research lifecycle, including developing a research idea, designing a study, storing and analyzing collected data, and writing and publishing reports or papers.

The [National Data Service](#) is an emerging vision for how scientists and researchers across all disciplines can find, reuse, and publish data.

[Galaxy Project](#) is an open, web-based platform for accessible, reproducible, and transparent computational biomedical research.

[Science Gateways Community Institute](#) allow science & engineering communities to access shared data, software, computing services, instruments, educational materials, and other resources specific to their disciplines.

1.30.2 Domain Specific Cyberinfrastructures

[HydroShare](#)

[OpenTopography](#)

[CyberGIS](#)

[Molecular Sciences Software Institute](#)

[Advanced Cyberinfrastructure Development \(ACID\)](#)

1.30.3 Funding Opportunities

National Science Foundation

[Cyberinfrastructure for Sustained Scientific Innovation \(CSSI\)](#)

[Cyberinfrastructure for Biological Research \(CIBR\)](#)

[Harnessing the Data Revolution \(HDR\): Institutes for Data-Intensive Research in Science and Engineering - Ideas Labs \(I-DIRSE-IL\)](#)

National Institutes of Health

Office of Cyber Infrastructure and Computational Biology (OCICB)

Bioinformatics

Other US Federal

Department of Energy ARPA-E

Department of Agriculture FACT

NOAA

Foundations

Alfred P. Sloan

Bill & Melinda Gates

Belmont Forum

Big 3

Google

Microsoft

Amazon

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



1.31 Introduction to containers

1.31.1 What is a container?

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A container image includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Why use containers?

- **Flexible:** Even the most complex applications can be containerized.
- **Lightweight:** Containers leverage and share the host kernel, making them much more efficient in terms of system resources than virtual machines.
- **Portable:** You can build locally, deploy to the cloud, and run anywhere.
- **Loosely coupled:** Containers are highly self sufficient and encapsulated, allowing you to replace or upgrade one without disrupting others.
- **Scalable:** You can increase and automatically distribute container replicas across a datacenter.
- **Secure:** Containers apply aggressive constraints and isolations to processes without any configuration required on the part of the user.

We can move and scale our containerized applications; containers effectively guarantee that those applications will run the same way anywhere, allowing us to quickly and easily take advantage of all these environments.

1.31.2 Working with containers

- **Image:** self-contained, read-only ‘snapshot’ of your applications and packages, with all their dependencies
- **Container:** A virtualization of an operating system run within an isolated user space. A running instance of an image.



Docker is a platform to build, share, and run applications with containers. Docker Engine is available on a variety of , and through Docker Desktop.

Docker images are built from Dockerfiles. A **Dockerfile** is a text document that contains all the commands you would normally execute manually in order to build a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile.

Once you have a Docker image you can:

- run it as a container (anywhere)
- push it to a registry (make it available to others)
- link it to GitHub with automated builds

Other things to note about Docker:

- Docker **always runs as root**. This makes it unsuitable for use on large computing systems with many users such as HPC.
- Docker images and containers are **stored in the Docker directory** (where Docker is installed) so you won't see them in a list of your files. There are special Docker commands you can use to list/remove them.



Singularity was created to run complex applications on HPC clusters in a simple, portable, and reproducible way. You are the **same user inside a container as outside**, and cannot gain additional privilege on the host system by default.

Singularity images are built from [definition files](#). Like Dockerfiles, they provide a list of commands necessary to build the image. They also have a very specific format although it is a different format from Dockerfiles.

Fortunately, Singularity will automatically convert and run Docker images (so you may not need to learn how to build a Singularity image at all).

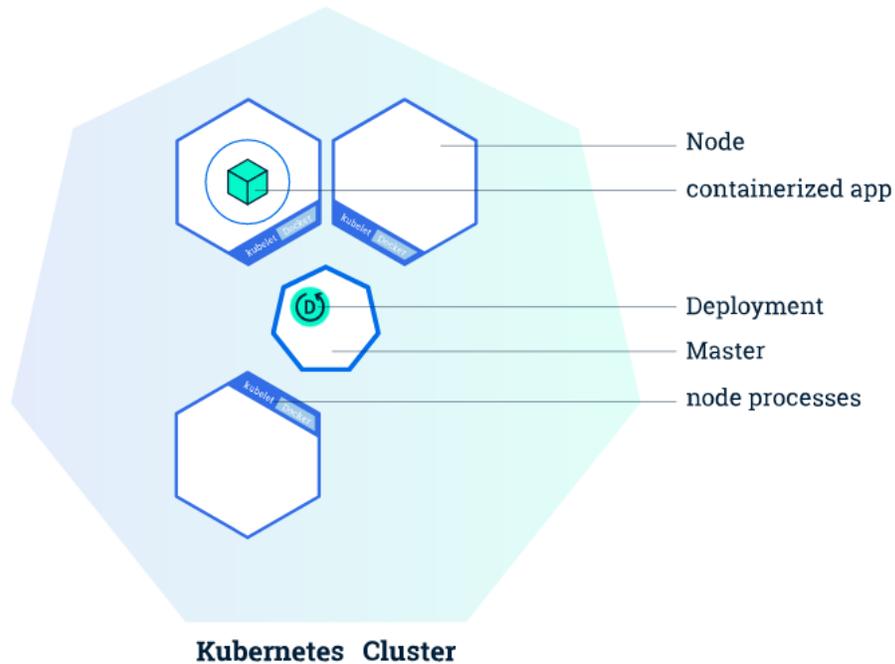
Once built, the Singularity image will be saved as a .sif file in your local working directory. You can easily see your image when you list your files but you may have images files saved to lots of different directories.



Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way. This allows you to scale-up your analyses as necessary.

A Kubernetes cluster consists of two types of resources:

- **master node:** responsible for deciding what runs on all of the cluster's nodes. This can include scheduling workloads, like containerized applications, and managing the workloads' lifecycle, scaling, and upgrades. The master also manages network and storage resources for those workloads.
- **worker node:** A cluster typically has one or more nodes, which are the worker machines that run your containerized applications and other workloads. Each node is managed from the master, which receives updates on each node's self-reported status.



More worker nodes = more compute power. This means you can easily scale your app to run much faster/with larger datasets.

Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. This provides a self-healing mechanism to address machine failure or maintenance.

1.31.3 Finding pre-built images

Image registry: a storage and content delivery system, such as that used by Docker

Warning: Only use images from trusted sources or images for which you can see the Dockerfile. An image from an untrusted source could contain something other than what it's labeled (eg. malware). If you can see the Dockerfile you can see exactly what is in the image.

Docker Hub is a service provided by Docker for finding and sharing container images with your team. It provides the following major features:

- **Repositories:** Push and pull container images.
- **Teams & Organizations:** Manage access to private repositories of container images.
- **Official Images:** Pull and use high-quality container images provided by Docker.
- **Publisher Images:** Pull and use high-quality container images provided by external vendors. Certified images also include support and guarantee compatibility with Docker Enterprise.
- **Builds:** Automatically build container images from GitHub and Bitbucket and push them to Docker Hub.

- **Webhooks:** Trigger actions after a successful push to a repository to integrate Docker Hub with other services.

Docker Hub is the most well-known and popular image registry for Docker containers.



BioContainers is a community-driven project that provides the infrastructure and basic guidelines to create, manage and distribute bioinformatics containers with **special focus in proteomics, genomics, transcriptomics and metabolomics**. BioContainers is based on the popular frameworks of Docker.

Although anyone can create a BioContainer, the majority of BioContainers are created by the Bioconda project. Every Bioconda package has a corresponding BioContainer available at Quay.io.

BIOCONDA[®]

- **package manager:** collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner
- is a channel for the conda package manager specializing in bioinformatics software. It consists of:
 - Over 800 contributors that add, modify, update and maintain the recipes
 - A repository of > 7000 bioinformatics packages ready to use
- **Each package added to Bioconda also has a corresponding Docker BioContainer automatically created and uploaded to Quay.io**
- You can [contribute](#) to the Bioconda project by building your own packages. Each package will also be made available as a BioContainer at

Note: The BioContainers registry search returns partial matches and matches to the tool description. So, if you want to find all the tools relevant to Nanopore analysis you can search for 'nanopore'.

Note: You want the docker images, not the Conda packages. Conda packages are not containers.

Quay is another general image registry. It works the same way as Docker Hub. However, Quay is home to all BioContainers made by the Bioconda project. Now we will find a BioContainer image at Quay, pull that image and run it on cloud virtual machine.

1.31.4 Hands-on

To run your BioContainer you will need a computer with Docker installed.

Launch this Atmosphere instance: **Ubuntu 18.04 GUI XFCE Base**

How to install Docker

Installing Docker on your computer takes a little time but it is reasonably straight forward and it is a one-time setup. . Docker installation is much easier on an Atmosphere instance with the ‘ezd’ command.

```
$ ezd
```

Get data to use with your container

```
$ cd Desktop
$ iget /iplant/home/shared/iplantcollaborative/example_data/porechop/SRR6059710.fastq
```

Use ‘docker pull’ to get the image

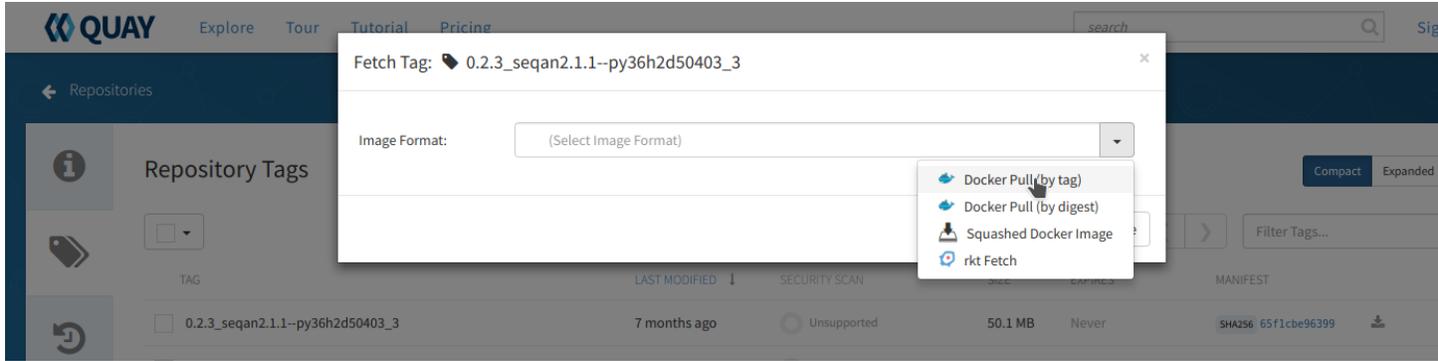
Go to and search for ‘porechop’ in the search bar at the top of the page.

Click on the ‘tag’ icon on the left side of the screen to show all the available ‘porechop’ images.

The screenshot shows the Quay.io web interface. At the top, there is a search bar and navigation links for 'Explore', 'Tour', 'Tutorial', and 'Pricing'. The main header indicates the current repository is 'biocontainers / porechop'. On the left sidebar, the 'Repository Tags' view is selected. The main content area displays a table of repository tags with columns for TAG, LAST MODIFIED, SECURITY SCAN, SIZE, EXPIRES, and MANIFEST. A 'Fetch Tag' button is visible on the right side of the table.

TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
<input type="checkbox"/> 0.2.3_seqan2.1.1--py36h2d50403_3	7 months ago	Unsupported	50.1 MB	Never	SHA256: 65f1cbe96399
<input type="checkbox"/> 0.2.3_seqan2.1.1--py35h2d50403_3	7 months ago	Unsupported	49.3 MB	Never	SHA256: 8ea41e99346a
<input type="checkbox"/> 0.2.3_seqan2.1.1--py35_2	a year ago	Unsupported	47.8 MB	Never	SHA256: 5c8483963e69
<input type="checkbox"/> 0.2.3_seqan2.1.1--py36_2	a year ago	Unsupported	48.4 MB	Never	SHA256: 3fb07c7cb2451
<input type="checkbox"/> 0.2.3_seqan2.1.1--py36_1	a year ago	Unsupported	48.4 MB	Never	SHA256: 1e3f4794cc59
<input type="checkbox"/> 0.2.3_seqan2.1.1--py35_1	a year ago	Unsupported	47.8 MB	Never	SHA256: 22504bb6deae
<input type="checkbox"/> 0.2.3_seqan2.1.1--0	a year ago	Unsupported	48.4 MB	Never	SHA256: 4bf6abaebe85

Click the ‘fetch tag’ icon at the right and choose ‘Docker pull (by tag)’ from the dropdown. This will copy the docker pull command that we will need on the command line.



Now you will need to pull the image from the registry onto your computer. Use the ‘docker pull’ command you copied from the registry above.

Note: If you are working on a system for which you don’t have root permissions you will need to use ‘sudo’ and provide your password. Like this:

```
$ sudo docker pull quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3
```

```
[amcooksey@rogue ~]$ sudo docker pull quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3
[sudo] password for amcooksey:
0.2.3_seqan2.1.1--py36h2d50403_3: Pulling from biocontainers/porechop
a3ed95caeb02: Already exists
b0dc45cd432d: Already exists
9466b3513669: Already exists
ddd482ea7b54: Already exists
4d69f833b9d8: Already exists
e7c454e5167d: Already exists
e38092b005c0: Already exists
f879b42dfe2b: Already exists
9417599398f7: Pull complete
Digest: sha256:65f1cbe96399eff89df55169f25d2b52f46115f9d4080c388fdeb7b22dc76b30
Status: Downloaded newer image for quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3
```

Use the ‘docker run’ command to run the container

The easiest way to test the container to run the help command for the tool. In this case ‘-h’ is the help command.

```
$ sudo docker run --rm -v $(pwd) :/working-dir -w /working-dir --entrypoint="porechop" \
↳ quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3 -h
```

From the result we are able to see the only required option is ‘-i INPUT’. Options in [square brackets] are not required.

Now we can run the container with our data file to see the output.

```
$ sudo docker run --rm -v $(pwd) :/working-dir -w /working-dir --entrypoint="porechop" \
↳ quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3 -i SRR6059710.fastq \
↳ -o porechop_output.fastq
```

We can break the command down into pieces so it is easier to read (the backslash represents where we have broken the line).

```
sudo \  
docker run \  
--rm \  
-v $(pwd) :/working-dir \  
-w /working-dir \  
--entrypoint="porechop" \  
quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3 \  
-i SRR6059710.fastq \  
-o porechop_out.fastq
```

What it All Means

- ‘sudo’ allows you to run the container with ‘root’ permissions—only required if you don’t have root permissions on your machine
- ‘docker run’ tells docker to run the container
- ‘–rm’ removes the container (not the image) from your system when the analysis is complete
- ‘-v’ mounts a *local* directory into a directory *within the container*
- ‘-w’ specifies the working directory within the container
- ‘–entrypoint’ tells the container what to do (usually the name of the tool; the command you would use to run the tool on the command line)
- ‘quay.io/biocontainers/porechop:0.2.3_seqan2.1.1--py36h2d50403_3’ is the name of the image we pulled from Quay.io
- ‘-i’ is the argument for the input file (FASTQ) for Porechop
- ‘-o’ is the argument for the output file (trimmed FASTQ) for Porechop

Important: You must supply an entrypoint on the command line when you run a BioContainer. It is possible to build entrypoints into a container but that is not the case with BioContainers.

```
Loading reads  
SRR6059710.fastq  
543,374 reads loaded  
↵  
  
Looking for known adapter sets  
4,990 / 10,000 (49.9%) █
```

Trimming adapters from read ends

```

SQK-NSK007_Y_Top: AATGTACTTCGTTCAGTTACGTATTGCT
SQK-NSK007_Y_Bottom: GCAATACGTAACTGAACGAAGT
SQK-MAP006_Y_Top_SK63: GGTTGTTTCTGTTGGTGCTGATATTGCT
SQK-MAP006_Y_Bottom_SK64: GCAATATCAGCACCAACAGAAA
PCR_1_start: ACTTGCCTGTCGCTCTATCTTC
PCR_1_end: GAAGATAGAGCGACAGGCAAGT
PCR_tail_1_start: TTAACCTTTCTGTTGGTGCTGATATTGC
PCR_tail_1_end: GCAATATCAGCACCAACAGAAAGGTTAA
PCR_tail_2_start: TTAACCTACTTGCCTGTCGCTCTATCTTC
PCR_tail_2_end: GAAGATAGAGCGACAGGCAAGTAGGTTAA

```

No adapters found - output reads are unchanged from input reads

Saving trimmed reads to file

Saved result to /working-dir/porechop_out.fastq

The output from Porechop is saved into the working directory within the container. We ran the container we mounted our current *local* working directory into the working directory *within the container*. The analysis has finished, the container has been removed (remember `-rm`) and now we should find our outputs in our *local* current working directory.

List the files:

```
$ ls -l
```

```

[amcooksey@rogue racon]$ ls -l
total 11350140
-rw-r----- 1 amcooksey iplant-everyone 346188054 May  8  2018 concat_reads.fastq
-rw-r----- 1 amcooksey iplant-everyone   23424 May  8  2018 miniasm_cat_output.fas
-rw-r----- 1 amcooksey iplant-everyone   11745 May  8  2018 minimap_cat_rnd2_out.p
-rw-r--r--  1 root      root          838803132 Feb 14 12:23 porechop_out.fastq
-rw-r--r--  1 amcooksey iplant-everyone 9579801472 May 14  2018 SRR6059708.fastq
-rw-r--r--  1 amcooksey iplant-everyone 857704006 May 14  2018 SRR6059710.fastq

```

You can see the ‘porechop_out.fastq’ file is in our current working directory. Notice that the this file is owned by ‘root’. This is because Docker containers always run as ‘root’.

At this point you can run your container on any system with Docker installed. To use this container on an HPC system you will need to use Singularity (rather than Docker) to run your container. For more information about running Docker containers with Singularity see the [Singularity documentation](#)

Useful Links

- [BioContainers](#)
- [Bioconda](#)
- [Request a BioContainer](#)
- [Singularity documentation](#)
- [BioContainers contribution guidelines](#)
- [Report BioContainers problems](#)

Some examples of public/private registries to consider for your research needs:

- Docker Cloud
- Docker Hub
- Docker Trusted Registry
- Amazon Elastic Container Registry
- Google Container Registry
- Azure Container Registry
- NVIDIA GPU Cloud
- Private Docker Registry - not official Docker
- Gitlab Container Registry
- Quay
- TreeScale
- Canister
- BioContainers Registry

Fix or improve this documentation:

- On Github:
- Send feedback: Tutorials@CyVerse.org



 [Learning Center Home](#)

1.32 Launching a Docker app on Atmosphere

You don't need to do the first 5 steps if you are running the 'DataCarpentry Genomics May2019' image.

1. Use web shell to run a basic Atmosphere instance*
2. Type `ezd`
3. Wait for Docker to install
4. Close or Refresh the Web Shell browser tab.
5. type `docker run hello-world`
6. type `docker run godlovedc/lolcow`

Note: You may receive an error if Docker did not add your username to the `docker` group, you'll need to use the `sudo` invocation, e.g. `sudo docker run hello-world`

To add yourself to the `docker` group type `sudo usermod -aG docker $USER` and refresh your terminal window.

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-

1.33 Introduction to Docker



1.33.1 1. Prerequisites

There are no specific skills needed for this tutorial beyond a basic comfort with the command line and using a text editor. Prior experience in developing web applications will be helpful but is not required.

1.33.2 2. Docker Installation

Getting all the tooling setup on your computer can be a daunting task, but not with Docker. Getting Docker up and running on your favorite OS (Mac/Windows/Linux) is very easy.

The getting started guide on Docker has detailed instructions for setting up Docker on [Mac/Windows/Linux](#).

Note: If you're using Docker for Windows make sure you have [shared your drive](#).

If you're using an older version of Windows or MacOS you may need to use [Docker Machine](#) instead.

All commands work in either Bash or Powershell on Windows.

Note: Depending on how you've installed Docker on your system, you might see a `permission denied` error after running the above command. If you're on Linux, you may need to prefix your Docker commands with `sudo`. Alternatively to run `docker` command without `sudo`, you need to add your user (who has root privileges) to `docker` group. For this run:

Create the `docker` group:

```
$ sudo groupadd docker
```

Add your user to the docker group:

```
$ sudo usermod -aG docker $USER
```

Log out and log back in so that your group membership is re-evaluated

2.1 Testing Docker installation

Once you are done installing Docker, test your Docker installation by running the following command to make sure you are using version 1.13 or higher:

```
$ docker --version
Docker version 18.09.3, build 774a1f4
```

When run without `--version` you should see a whole bunch of lines showing the different options available with `docker`. Alternatively you can test your installation by running the following:

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
03f4658f8b78: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca36966a7
Status: Downloaded newer image for hello-world:latest

Hello from Docker.
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
.....
```

1.33.3 3. Running Docker containers from prebuilt images

Now that you have everything setup, it's time to get our hands dirty. In this section, you are going to run a container from [Alpine Linux](#) (a lightweight linux distribution) image on your system and get a taste of the `docker run` command.

But wait, what exactly is a container and image?

Containers - Running instances of Docker images — containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS.

Images - The file system and configuration of our application which are used to create containers. To find out more about a Docker image, run `docker inspect hello-world`. In the demo above, you could have used the `docker pull` command to download the `hello-world` image. However when you executed the command `docker run hello-world`, it also did a `docker pull` behind the scenes to download the `hello-world` image with `latest` tag (we will learn more about tags little later).

Now that we know what a container and image is, let's run the following command in our terminal:

```
$ docker run alpine ls -l
total 52
drwxr-xr-x    2 root    root          4096 Dec 26  2016 bin
drwxr-xr-x    5 root    root           340 Jan 28  09:52 dev
drwxr-xr-x   14 root    root          4096 Jan 28  09:52 etc
drwxr-xr-x    2 root    root          4096 Dec 26  2016 home
drwxr-xr-x    5 root    root          4096 Dec 26  2016 lib
drwxr-xr-x    5 root    root          4096 Dec 26  2016 media
.....
```

Similar to `docker run hello-world` command in the demo above, `docker run alpine ls -l` command fetches the `alpine:latest` image from the Docker registry first, saves it in our system and then runs a container from that saved image.

When you run `docker run alpine`, you provided a command `ls -l`, so Docker started the command specified and you saw the listing

You can use the `docker images` command to see a list of all images on your system

```
$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED           VIRTUAL SIZE
↪VIRTUAL SIZE
alpine              latest      c51f86c28340     4 weeks ago      1.109 MB
↪109 MB
hello-world        latest      690ed74de00f     5 months ago     960 B
↪960 B
```

Let's try something more exciting.

```
$ docker run alpine echo "Hello world"
Hello world
```

OK, that's some actual output. In this case, the Docker client dutifully ran the `echo` command in our `alpine` container and then exited it. If you've noticed, all of that happened pretty quickly. Imagine booting up a virtual machine, running a command and then killing it. Now you know why they say containers are fast!

Try another command.

```
$ docker run alpine sh
```

Wait, nothing happened! Is that a bug? Well, no. These interactive shells will exit after running any scripted commands such as `sh`, unless they are run in an interactive terminal - so for this example to not exit, you need to `docker run -it alpine sh`. You are now inside the container shell and you can try out a few commands like `ls -l`, `uname -a` and others.

Before doing that, now it's time to see the `docker ps` command which shows you all containers that are currently running.

```
$ docker ps
CONTAINER ID          IMAGE          COMMAND          CREATED           STATUS          PORTS
↪STATUS              PORTS

```

Since no containers are running, you see a blank line. Let's try a more useful variant: `docker ps -a`

```
$ docker ps -a
CONTAINER ID          IMAGE          PORTS          COMMAND          NAMES          CREATED           STATUS
↪STATUS              PORTS          NAMES
36171a5da744        alpine        "/bin/sh"      5 minutes ago    fervent_newton
↪Exited (0) 2 minutes ago
```

(continues on next page)

(continued from previous page)

a6a9d46d0b2f	alpine	"echo 'hello from alp"	6 minutes ago	↳
↳ Exited (0) 6 minutes ago		lonely_kilby		
ff0a5c3750b9	alpine	"ls -l"	8 minutes ago	↳
↳ Exited (0) 8 minutes ago		elated_ramanujan		
c317d0a9e3d2	hello-world	"/hello"	34 seconds ago	↳
↳ Exited (0) 12 minutes ago		stupefied_mcclintock		

What you see above is a list of all containers that you ran. Notice that the STATUS column shows that these containers exited a few minutes ago.

If you want to run scripted commands such as `sh`, they should be run in an interactive terminal. In addition, interactive terminal allows you to run more than one command in a container. Let's try that now:

```
$ docker run -it alpine sh
/ # ls
bin    dev    etc    home  lib    media mnt    proc  root  run    sbin  srv    ↳
↳ sys  tmp    usr    var
/ # uname -a
Linux de4bbc3eeaec 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 Linux
```

Running the `run` command with the `-it` flags attaches us to an interactive `tty` in the container. Now you can run as many commands in the container as you want. Take some time to run your favorite commands.

Exit out of the container by giving the `exit` command.

```
/ # exit
```

Note: If you type `exit` your **container** will exit and is no longer active. To check that, try the following:

```
$ docker ps -l
CONTAINER ID        IMAGE               PORTS              COMMAND              CREATED             NAMES
↳ STATUS
de4bbc3eeaec       alpine             "/bin/sh"         3 minutes ago       pensive_leavitt
↳ Exited (0) About a minute ago
```

If you want to keep the container active, then you can use keys `ctrl +p`, `ctrl +q`. To make sure that it is not exited run the same `docker ps -a` command again:

```
$ docker ps -l
CONTAINER ID        IMAGE               PORTS              COMMAND              CREATED             NAMES
↳ STATUS
0db38ea51a48       alpine             "sh"              3 minutes ago       elastic_lewin
↳ Up 3 minutes
```

Now if you want to get back into that container, then you can type `docker attach <container id>`. This way you can save your container:

```
$ docker attach 0db38ea51a48
```

1.33.4 4. Build Docker images which contain your own code

Great! so you have now looked at `docker run`, played with a Docker containers and also got the hang of some terminology. Armed with all this knowledge, you are now ready to get to the real stuff — deploying your own applications with Docker.

4.1 Deploying a command-line app

Note: Code for this section is in this repo in the `examples/` directory

In this section, let's dive deeper into what Docker images are. Later on we will build our own image and use that image to run an application locally.

4.1.1 Docker images

Docker images are the basis of containers. In the previous example, you pulled the `alpine` image from the registry and asked the Docker client to run a container based on that image. To see the list of images that are available locally on your system, run the `docker images` command.

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              bionic             47b19964fb50      4 weeks ago       88.1MB
alpine              latest             caf27325b298      4 weeks ago       5.53MB
hello-world        latest             fce289e99eb9      2 months ago      1.84kB
.....
```

Above is a list of images that I've pulled from the registry and those I've created myself (we'll shortly see how). You will have a different list of images on your machine. The **TAG** refers to a particular snapshot of the image and the **ID** is the corresponding unique identifier for that image.

For simplicity, you can think of an image akin to a git repository - images can be committed with changes and have multiple versions. When you do not provide a specific version number, the client defaults to latest.

For example you could pull a specific version of ubuntu image as follows:

```
$ docker pull ubuntu:16.04
```

If you do not specify the version number of the image, as mentioned, the Docker client will default to a version named latest.

So for example, the `docker pull` command given below will pull an image named `ubuntu:latest`

```
$ docker pull ubuntu
```

To get a new Docker image you can either get it from a registry (such as the Docker hub) or create your own. There are hundreds of thousands of images available on Docker hub. You can also search for images directly from the command line using `docker search`.

```
$ docker search ubuntu
NAME                STARS              OFFICIAL    DESCRIPTION                                     AUTOMATED
ubuntu              7310              [OK]       Ubuntu is a Debian-based Linux operating sys...
dorowu/ubuntu-desk... 163               [OK]       Ubuntu with openssh-server and NoVNC
rastasheep/ubuntu-sshd 131              [OK]       Dockerized SSH service, built on top of offi...
[OK]
```

(continues on next page)

(continued from previous page)

ansible/ubuntu14.04-ansible		Ubuntu 14.04 LTS with
↪ansible	90	[OK]
ubuntu-upstart		Upstart is an event-based
↪replacement for th...	81	[OK]
neurodebian		NeuroDebian provides
↪neuroscience research s...	43	[OK]
ubuntu-debootstrap		debootstrap --
↪variant=minbase --components=m...	35	[OK]
landlinternet/ubuntu-16-nginx-php-phpmyadmin-mysql-5		ubuntu-16-nginx-php-
↪phpmyadmin-mysql-5	26	[OK]
nuagebec/ubuntu		Simple always updated Ubuntu
↪docker images w...	22	[OK]
tutum/ubuntu		Simple Ubuntu docker images
↪with SSH access	18	
ppc64le/ubuntu		Ubuntu is a Debian-based
↪Linux operating sys...	11	
i386/ubuntu		Ubuntu is a Debian-based
↪Linux operating sys...	9	
landlinternet/ubuntu-16-apache-php-7.0		ubuntu-16-apache-php-7.0
↪	7	[OK]
eclipse/ubuntu_jdk8		Ubuntu, JDK8, Maven 3, git,
↪curl, nmap, mc, ...	5	[OK]
darksheer/ubuntu		Base Ubuntu Image -- Updated
↪hourly	3	[OK]
codenvy/ubuntu_jdk8		Ubuntu, JDK8, Maven 3, git,
↪curl, nmap, mc, ...	3	[OK]
landlinternet/ubuntu-16-nginx-php-5.6-wordpress-4		ubuntu-16-nginx-php-5.6-
↪wordpress-4	2	[OK]
landlinternet/ubuntu-16-nginx		ubuntu-16-nginx
↪	2	[OK]
pivotaldata/ubuntu		A quick freshening-up of the
↪base Ubuntu doc...	1	
smartentry/ubuntu		ubuntu with smartentry
↪	0	[OK]
pivotaldata/ubuntu-gpdb-dev		Ubuntu images for GPDB
↪development	0	
landlinternet/ubuntu-16-healthcheck		ubuntu-16-healthcheck
↪	0	[OK]
thatsamguy/ubuntu-build-image		Docker webapp build images
↪based on Ubuntu	0	
ossobv/ubuntu		Custom ubuntu image from
↪scratch (based on o...	0	
landlinternet/ubuntu-16-sshd		ubuntu-16-sshd
↪	0	[OK]

An important distinction with regard to images is between base images and child images and official images and user images (Both of which can be base images or child images.).

Important: **Base images** are images that have no parent images, usually images with an OS like ubuntu, alpine or debian.

Child images are images that build on base images and add additional functionality.

Official images are Docker sanctioned images. Docker, Inc. sponsors a dedicated team that is responsible for reviewing and publishing all Official Repositories content. This team works in collaboration with upstream software maintainers, security experts, and the broader Docker community. These are not prefixed by an organization or user name. In the list of images above, the python, node, alpine and nginx images are official (base) images. To find out

more about them, check out the Official Images Documentation.

User images are images created and shared by users like you. They build on base images and add additional functionality. Typically these are formatted as `user/image-name`. The user value in the image name is your Dockerhub user or organization name.

4.1.2 Meet our Python app

Now that you have a better understanding of images, it's time to create an image that sandboxes a small Python application. We'll do this by creating a small Python script which prints a welcome message, then dockerizing it by writing a Dockerfile, and finally we'll build the image and run it.

- Create a Python script
- Build the image
- Run your image

1. Create a Python script which prints a welcome message

Start by creating a directory called `simple-script` where we'll create the following files:

- `app.py`
- Dockerfile

```
$ mkdir simple-script && cd simple-script
```

1.1 `app.py`

Create the `app.py` file with the following content. You can use any of favorite text editor to create this file.

```
print('hello world!')
print('this is my first attempt')
```

Note: If you want, you can run this app through your laptop's native Python installation first just to see what it looks like. Run `python app.py`.

You should see the message:

```
hello world!this is my first attempt
```

This is totally optional - but some people like to see what the app's supposed to do before they try to Dockerize it.

1.2. Dockerfile

A **Dockerfile** is a text file that contains a list of commands that the Docker daemon calls while creating an image. The Dockerfile contains all the information that Docker needs to know to run the app — a base Docker image to run from, location of your project code, any dependencies it has, and what commands to run at start-up. It is a simple way to automate the image creation process. The best part is that the commands you write in a Dockerfile are almost identical to their equivalent Linux commands. This means you don't really have to learn new syntax to create your own Dockerfiles.

We want to create a Docker image with this app. As mentioned above, all user images are based on a base image. Since our application is written in Python, we will build our own Python image based on `Alpine`. We'll do that using a Dockerfile.

Create a file called `Dockerfile` in the `simple-script` directory, and add content to it as described below.

```
# our base image# our base image
FROM alpine:3.9

# install python and pip
RUN apk add --update py3-pip

# copy files required for the app to run
COPY app.py /usr/src/app/

# run the application
CMD python3 /usr/src/app/app.py
```

Now let's see what each of those lines mean..

1.2.1 We'll start by specifying our base image, using the FROM keyword:

```
FROM alpine:3.9
```

1.2.2. The next step usually is to write the commands of copying the files and installing the dependencies. But first we will install the Python pip package to the alpine linux distribution. This will not just install the pip package but any other dependencies too, which includes the python interpreter. Add the following RUN command next:

```
RUN apk add --update py3-pip
```

1.2.3. Copy the file you have created earlier into our image by using COPY command.

```
COPY app.py /usr/src/app/
```

1.2.4. The last step is the command for running the application. Use the CMD command to do that:

```
CMD python3 /usr/src/app/app.py
```

The primary purpose of CMD is to tell the container which command it should run by default when it is started.

2. Build the image

Now that you have your Dockerfile, you can build your image. The `docker build` command does the heavy-lifting of creating a docker image from a Dockerfile.

The `docker build` command is quite simple - it takes an optional tag name with the `-t` flag, and the location of the directory containing the Dockerfile - the `.` indicates the current directory:

Note: When you run the `docker build` command given below, make sure to replace `<YOUR_DOCKERHUB_USERNAME>` with your username. This username should be the same one you created when registering on Docker hub. If you haven't done that yet, please go ahead and create an account in [Dockerhub](#).

```
YOUR_DOCKERHUB_USERNAME=<YOUR_DOCKERHUB_USERNAME>
```

For example this is how I assign my dockerhub username

```
YOUR_DOCKERHUB_USERNAME=jpistorius
```

Now build the image using the following command:

```
$ docker build -t $YOUR_DOCKERHUB_USERNAME/simple-script .
Sending build context to Docker daemon 10.24kB
```

(continues on next page)

(continued from previous page)

```
Step 1/4 : FROM alpine:3.9
---> caf27325b298
Step 2/4 : RUN apk add --update py3-pip
---> Using cache
---> dad2a197fcad
Step 3/4 : COPY app.py /usr/src/app/
---> Using cache
---> a8ebf6cd2735
Step 4/4 : CMD python3 /usr/src/app/app.py
---> Using cache
---> a1fb2906a937
Successfully built a1fb2906a937
Successfully tagged jpistorius/simple-script:latest
```

If you don't have the `alpine:3.9` image, the client will first pull the image and then create your image. Therefore, your output on running the command will look different from mine. If everything went well, your image should be ready! Run `docker images` and see if your image `$YOUR_DOCKERHUB_USERNAME/simple-script` shows.

3. Run your image

When Docker can successfully build your Dockerfile, test it by starting a new container from your new image using the `docker run` command.

```
$ docker run $YOUR_DOCKERHUB_USERNAME/simple-script
```

You should see something like this:

```
hello world!
this is my first attempt
```

4.2 Deploying a Jupyter Notebook

In this section, let's build a Docker image which can run a Jupyter Notebook

4.2.1 Suitable Docker images for a base

Search for images on Docker Hub which contain the string 'jupyter'

```
$ docker search jupyter
NAME                                DESCRIPTION
↔ STARS                            OFFICIAL    AUTOMATED
jupyter/datascience-notebook      Jupyter Notebook Data Science Stack from
↔htt... 446
jupyter/all-spark-notebook         Jupyter Notebook Python, Scala, R, Spark,
↔Me... 223
jupyterhub/jupyterhub              JupyterHub: multi-user Jupyter notebook
↔serv... 195 [OK]
jupyter/scipy-notebook             Jupyter Notebook Scientific Python Stack
↔fro... 155
jupyter/tensorflow-notebook       Jupyter Notebook Scientific Python Stack w/
↔... 116
jupyter/pyspark-notebook          Jupyter Notebook Python, Spark, Mesos Stack
↔... 95
jupyter/minimal-notebook          Minimal Jupyter Notebook Stack from https://
↔... 73
```

(continues on next page)

(continued from previous page)

ermaker/keras-jupyter ↪... 66	Jupyter with Keras (with Theano backend and [OK]
jupyter/base-notebook ↪stacks... 60	Small base image for Jupyter Notebook
xblaster/tensorflow-jupyter ↪ 52	Dockerized Jupyter with tensorflow [OK]
jupyter/r-notebook ↪github... 22	Jupyter Notebook R Stack from https://
jupyterhub/singleuser ↪Jupyter... 21	single-user docker images for use with [OK]
...	

4.2.2 Meet our model

Let's deploy a Python function inside a Docker image along with Jupyter.

- *Create a Python file containing a function*
- **'Build the image'**
- **'Run your image'**

1. Create a Python file containing a function

Start by creating a directory called `myfirstapp` where we'll create the following files:

- `model.py`
- `Dockerfile`

```
$ mkdir myfirstapp && cd myfirstapp
```

1.1 model.py

Create the `model.py` file with the following content. You can use any of favorite text editor to create this file.

```
def introduce(name):
    return 'Hello ' + name
```

1.2. Dockerfile

Since we want to use a Jupyter notebook to call our function, we will build an image based on `jupyter/minimal-notebook`.

Note: This is one of the official Docker images provided by the Jupyter project for you to build your own data science notebooks on:

<https://jupyter-docker-stacks.readthedocs.io/en/latest/>

Create a file called `Dockerfile` in the `myfirstapp` directory, and add content to it as described below.

```
# our base image
FROM jupyter/minimal-notebook

# copy files required for the model to work
COPY model.py /home/jovyan/work/
```

(continues on next page)

(continued from previous page)

```
# tell the port number the container should expose
EXPOSE 8888
```

Now let's see what each of those lines mean..

1.2.1 We'll start by specifying our base image, using the FROM keyword:

```
FROM jupyter/minimal-notebook
```

1.2.2. Copy the file you have created earlier into our image by using COPY command.

```
COPY model.py /home/jovyan/work/
```

1.2.3. Specify the port number which needs to be exposed. Since Jupyter runs on 8888 that's what we'll expose.

```
EXPOSE 8888
```

1.2.4. What about CMD?

Notice that unlike our previous Dockerfile this one does not end with a CMD command. This is on purpose.

Remember: The primary purpose of CMD is to tell the container which command it should run by default when it is started.

Can you guess what will happen if we don't specify our own 'entrypoint' using CMD?

2. Build the image

Note: Remember to replace <YOUR_DOCKERHUB_USERNAME> with your username. This username should be the same one you created when registering on Docker hub.

```
YOUR_DOCKERHUB_USERNAME=<YOUR_DOCKERHUB_USERNAME>
```

For example this is how I assign my dockerhub username

```
YOUR_DOCKERHUB_USERNAME=jpistorius
```

Now build the image using the following command:

```
$ docker build -t $YOUR_DOCKERHUB_USERNAME/myfirstapp .
Sending build context to Docker daemon 3.072kB
Step 1/3 : FROM jupyter/minimal-notebook
---> 36c8dd0e1d8f
Step 2/3 : COPY model.py /home/jovyan/work/
---> b61aefd7a735
Step 3/3 : EXPOSE 8888
---> Running in 519dcabe4eb3
Removing intermediate container 519dcabe4eb3
---> 7983fe164dc6
Successfully built 7983fe164dc6
Successfully tagged jpistorius/myfirstapp:latest
```

If everything went well, your image should be ready! Run docker images and see if your image \$YOUR_DOCKERHUB_USERNAME/myfirstapp shows.

3. Run your image

When Docker can successfully build your Dockerfile, test it by starting a new container from your new image using the `docker run` command. Don't forget to include the port forwarding options you learned about before.

```
$ docker run -p 8888:8888 $YOUR_DOCKERHUB_USERNAME/myfirstapp
```

You should see something like this:

```
Executing the command: jupyter notebook
[I 07:21:25.396 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.
↳local/share/jupyter/runtime/notebook_cookie_secret
[I 07:21:25.609 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.
↳7/site-packages/jupyterlab
[I 07:21:25.609 NotebookApp] JupyterLab application directory is /opt/conda/share/
↳jupyter/lab
[I 07:21:25.611 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 07:21:25.611 NotebookApp] The Jupyter Notebook is running at:
[I 07:21:25.611 NotebookApp] http://(29a022bb5807 or 127.0.0.1):8888/?token=copy-your-
↳own-token-not-this-one
[I 07:21:25.611 NotebookApp] Use Control-C to stop this server and shut down all
↳kernels (twice to skip confirmation).
[C 07:21:25.612 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
    http://(29a022bb5807 or 127.0.0.1):8888/?token=copy-your-own-token-not-this-
↳one
```

Head over to <http://localhost:8888> and your Jupyter notebook server should be running.

Note: Copy the token from your own `docker run` output and paste it into the 'Password or token' input box.

1.33.5 5. Dockerfile commands summary

Here's a quick summary of the few basic commands we used in our Dockerfiles.

- **FROM** starts the Dockerfile. It is a requirement that the Dockerfile must start with the FROM command. Images are created in layers, which means you can use another image as the base image for your own. The FROM command defines your base layer. As arguments, it takes the name of the image. Optionally, you can add the Dockerhub username of the maintainer and image version, in the format `username/image:version`.
- **RUN** is used to build up the Image you're creating. For each RUN command, Docker will run the command then create a new layer of the image. This way you can roll back your image to previous states easily. The syntax for a RUN instruction is to place the full text of the shell command after the RUN (e.g., `RUN mkdir /user/local/foo`). This will automatically run in a `/bin/sh` shell. You can define a different shell like this: `RUN /bin/bash -c 'mkdir /user/local/foo'`
- **COPY** copies local files into the container.
- **CMD** defines the commands that will run on the Image at start-up. Unlike a RUN, this does not create a new layer for the Image, but simply runs the command. There can only be one CMD per a Dockerfile/Image. If you need to run multiple commands, the best way to do that is to have the CMD run a script. CMD requires that you tell it where to run the command, unlike RUN. So example CMD commands would be:

```
CMD ["python", "./app.py"]
CMD ["/bin/bash", "echo", "Hello World"]
```

- EXPOSE creates a hint for users of an image which ports provide services. It is included in the information which can be retrieved via `$ docker inspect <container-id>`.

Note: The EXPOSE command does not actually make any ports accessible to the host! Instead, this requires publishing ports by means of the `-p` flag when using `docker run`.

- PUSH pushes your image to Docker Cloud, or alternately to a private registry

Note: If you want to learn more about Dockerfiles, check out [Best practices for writing Dockerfiles](#).

1.33.6 6. Demos

6.1 Portainer

Portainer is an open-source lightweight management UI which allows you to easily manage your Docker hosts or Swarm cluster.

- Simple to use: It has never been so easy to manage Docker. Portainer provides a detailed overview of Docker and allows you to manage containers, images, networks and volumes. It is also really easy to deploy, you are just one Docker command away from running Portainer anywhere.
- Made for Docker: Portainer is meant to be plugged on top of the Docker API. It has support for the latest versions of Docker, Docker Swarm and Swarm mode.

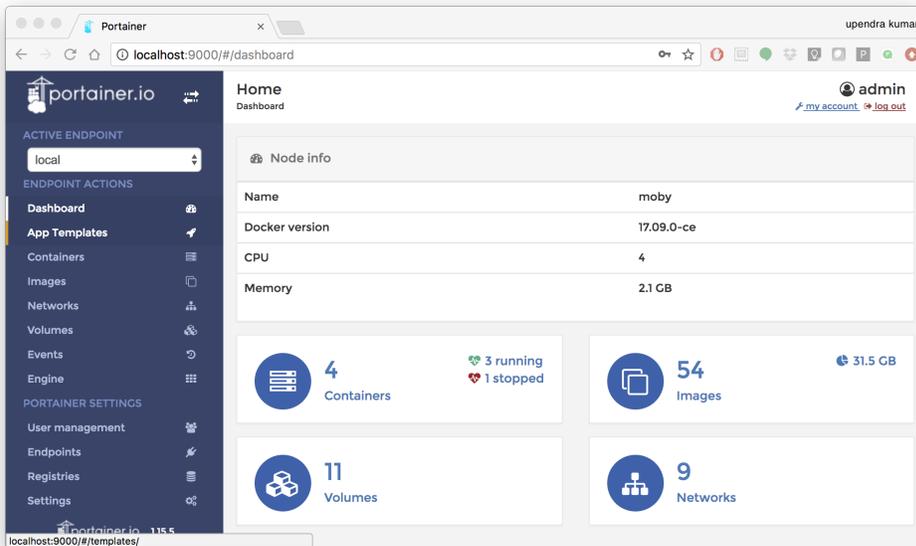
6.1.1 Installation

Use the following Docker commands to deploy Portainer. Now the second line of command should be familiar to you by now. We will talk about first line of command in the Advanced Docker session.

```
$ docker volume create portainer_data
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock -v ↵
↵portainer_data:/data portainer/portainer
```

- If you are on mac, you'll just need to access the port 9000 (<http://localhost:9000>) of the Docker engine where portainer is running using username `admin` and password `tryportainer`
- If you are running Docker on Atmosphere/Jetstream or on any other cloud, you can open `ipaddress:9000`. For my case this is `http://128.196.142.26:9000`

Note: The `-v /var/run/docker.sock:/var/run/docker.sock` option can be used in mac/linux environments only.



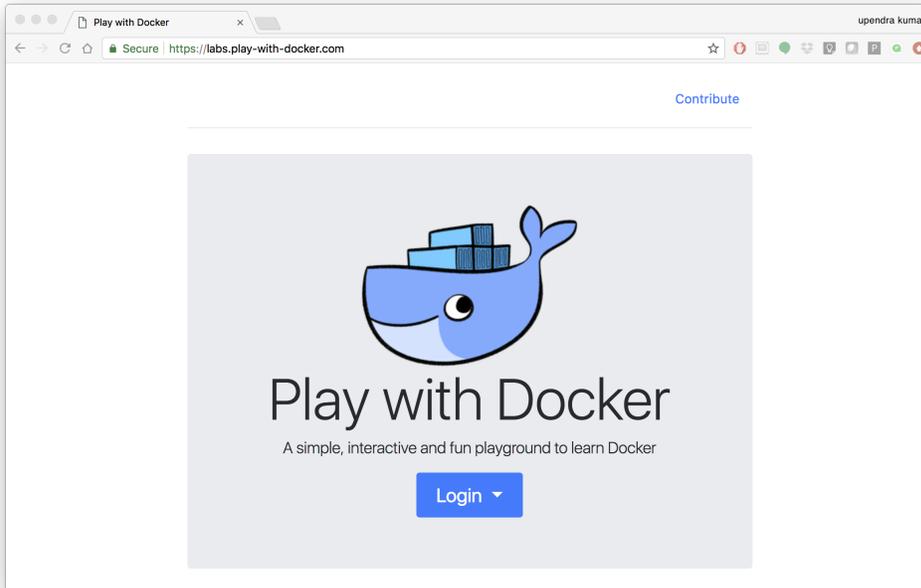
6.2 Play-with-docker (PWD)

PWD is a Docker playground which allows users to run Docker commands in a matter of seconds. It gives the experience of having a free Alpine Linux Virtual Machine in browser, where you can build and run Docker containers and even create clusters in [Docker Swarm Mode](#). Under the hood, Docker-in-Docker (DinD) is used to give the effect of multiple VMs/PCs. In addition to the playground, PWD also includes a training site composed of a large set of Docker labs and quizzes from beginner to advanced level available at training.play-with-docker.com.

6.2.1 Installation

You don't have to install anything to use PWD. Just open <https://labs.play-with-docker.com/> <<https://labs.play-with-docker.com/>> and start using PWD

Note: You can use your Dockerhub credentials to log-in to PWD



1.34 Advanced Docker

Now that we are *relatively* comfortable with Docker, let's look at some advanced Docker topics, such as:

- Registry
- Porting a Docker image to a Registry & Repository (public and private)
- Managing data within containers
- Deploying containers on cloud services

1.34.1 1. Docker Registries

To demonstrate the portability of what we just created, let's upload our built Docker image to a Docker Registry and then run it somewhere else (i.e. [CyVerse Atmosphere](#)).

In this exercise, you'll learn how to push built containers to registries, pull those containers from registries, and run those containers on remote hosts (virtual machines).

This will benefit you when you want to deploy new containers to production environments where testing is not possible.

Important: So what *EXACTLY* is a **Registry**?

A registry is a collection of Repositories, and a Repository is a collection of Images. A Docker Registry is sort of like a GitHub Repository, except the code is already compiled, in this case, into a container. You must have an account on a registry. You can create many repositories. The Docker CLI uses Docker's public registry by default. You can even set up your own private registry using Docker Trusted Registry

There are several things you can do with Docker registries:

- Push images

- Find images
- Pull images
- Share images

1.1 Popular Registries

Some examples of public/private registries to consider for your research needs:

- Docker Cloud
- Docker Hub
- Docker Trusted Registry
- Amazon Elastic Container Registry
- Google Container Registry
- Azure Container Registry
- NVIDIA GPU Cloud
- Private Docker Registry - not official Docker
- Gitlab Container Registry
- CoreOS Quay
- TreeScale
- Canister

1.1.1 Log into a Registry with your Docker ID

Now that you've created and tested your image, you can push it to Docker cloud or Docker hub.

Note: If you don't have an account, sign up for one at [Docker Cloud](#) or [Docker Hub](#). Make note of your username. There are several advantages of registering to DockerHub which we will see later on in the session

First, you have to login to your Docker Hub account.

If you want to authenticate to a different Registry, type the name of the registry after `login`:

```
$ docker login <registry-name>
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/tswetnam/.docker/config.
↪ json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

If it is your first time logging in you will be queried for your username and password.

Login with your Docker ID to push and pull images from Docker Hub or private registry.

If you don't have a Docker ID, head over to <https://hub.docker.com> to create one.

1.1.2 Tagging images

The notation for associating a local image with a repository on a registry is `username/repository:tag`. The tag is optional, but recommended, since it is the mechanism that registries use to give Docker images a version. Give the repository and tag meaningful names for the context, such as `get-started:part2`. This will put the image in the `get-started` repository and tag it as `part2`.

Note: By default the docker image gets a `latest` tag if you don't provide one. Thought convenient, it is not recommended for reproducibility purposes.

Now, put it all together to tag the image. Run `docker tag` image with your username, repository, and tag names so that the image will upload to your desired destination. For our docker image since we already have our Dockerhub username we will just add tag which in this case is `1.0`

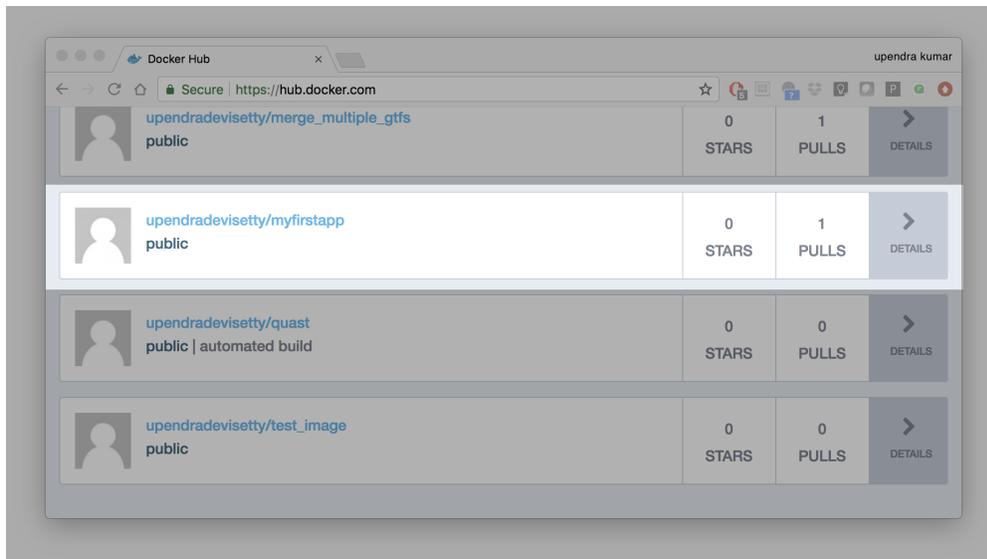
```
$ docker tag username/appname:latest username/appname:1.0
```

1.1.3 Publish the image

Upload your tagged image to the Dockerhub repository

```
$ docker push username/appname:1.0
```

Once complete, the results of this upload are publicly available. If you log in to Docker Hub, you will see the new image there, with its pull command.



Congrats! You just made your first Docker image and shared it with the world!

1.1.4 Pull and run the image from the remote repository

Let's try to run the image from the remote repository on Cloud server by logging into CyVerse Atmosphere, [launching an instance](#)

First install Docker on Atmosphere using from here <https://docs.docker.com/install/linux/docker-ce/ubuntu> or alternatively you can use `ezd` command which is a short-cut command for installing Docker on Atmosphere

```
$ ezd
```

Now run the following command to run the docker image from Dockerhub

```
$ sudo docker run -d -p 8888:8888 --name jupyter username/jupyter:1.0
```

Note: You don't have to run `docker pull` since if the image isn't available locally on the machine, Docker will pull it from the repository.

Head over to <http://<vm-address>:8888> and your app should be live.

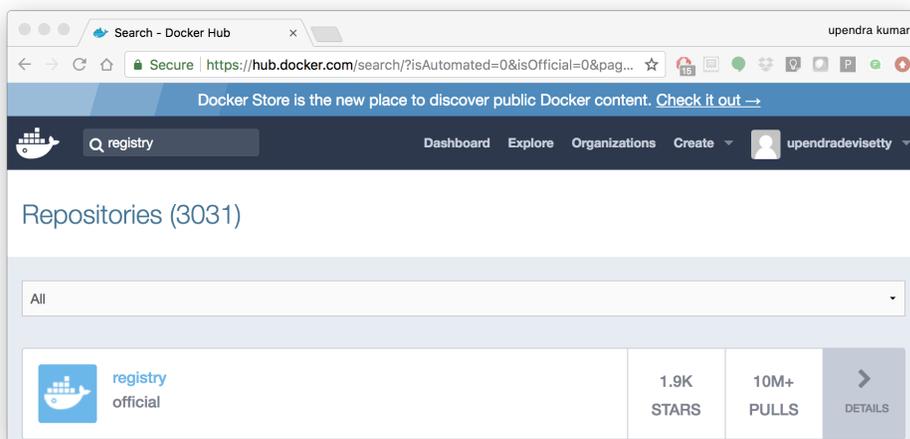
1.2 Private repositories

In an earlier part, we had looked at the Docker Hub, which is a public registry that is hosted by Docker. While the Dockerhub plays an important role in giving public visibility to your Docker images and for you to utilize quality Docker images put up by others, there is a clear need to setup your own private registry too for your team/organization. For example, CyVerse has its own private registry which will be used to push the Docker images.

1.2.1 Pull down the Registry Image

You might have guessed by now that the registry must be available as a Docker image from the Docker Hub and it should be as simple as pulling the image down and running that. You are correct!

A Dockerhub search on the keyword `registry` brings up the following image as the top result:



Run a container from `registry` Dockerhub image

```
$ docker run -d -p 5000:5000 --name registry registry:2
```

Run `docker ps -l` to check the recent container from this Docker image

```
$ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED
↳STATUS          PORTS              NAMES              11 seconds ago
6e44a0459373      registry:2        "/entrypoint.sh /e...
↳Up 10 seconds    0.0.0.0:5000->5000/tcp  registry
```

1.2.2 Tag the image that you want to push

Next step is to tag your image under the registry namespace and push it there

```
$ REGISTRY=localhost:5000
$ docker tag $YOUR_DOCKERHUB_USERNAME/myfirstapp:1.0 $REGISTRY/$(whoami)/myfirstapp:1.
↳0
```

1.2.2 Publish the image into the local registry

Finally push the image to the local registry

```
$ docker push $REGISTRY/$(whoami)/myfirstapp:1.0
The push refers to a repository [localhost:5000/upendra_35/myfirstapp]
64436820c85c: Pushed
831cff83ec9e: Pushed
c3497b2669a8: Pushed
1c5b16094682: Pushed
c52044a91867: Pushed
60ab55d3379d: Pushed
1.0: digest: sha256:5095dea8b2cf308c5866ef646a0e84d494a00ff0e9b2c8e8313a176424a230ce_
↳size: 1572
```

1.2.3 Pull and run the image from the local repository

You can also pull the image from the local repository similar to how you pull it from Dockerhub and run a container from it

```
$ docker run -d -P --name=myfirstapplocal $REGISTRY/$(whoami)/myfirstapp:1.0
```

1.34.2 2. Automated Docker image building from GitHub

An automated build is a Docker image build that is triggered by a code change in a GitHub or Bitbucket repository. By linking a remote code repository to a Dockerhub automated build repository, you can build a new Docker image every time a code change is pushed to your code repository.

A build context is a Dockerfile and any files at a specific location. For an automated build, the build context is a repository containing a Dockerfile.

Automated Builds have several advantages:

- Images built in this way are built exactly as specified.
- The Dockerfile is available to anyone with access to your Docker Hub repository.

- Your repository is kept up-to-date with code changes automatically.
- Automated Builds are supported for both public and private repositories on both GitHub and Bitbucket.

2.1 Prerequisites

To use automated builds, you first must have an account on [Docker Hub](#) and on the hosted repository provider ([GitHub](#) or [Bitbucket](#)). While Docker Hub supports linking both GitHub and Bitbucket repositories, here we will use a GitHub repository. If you don't already have one, make sure you have a GitHub account. A basic account is free

Note:

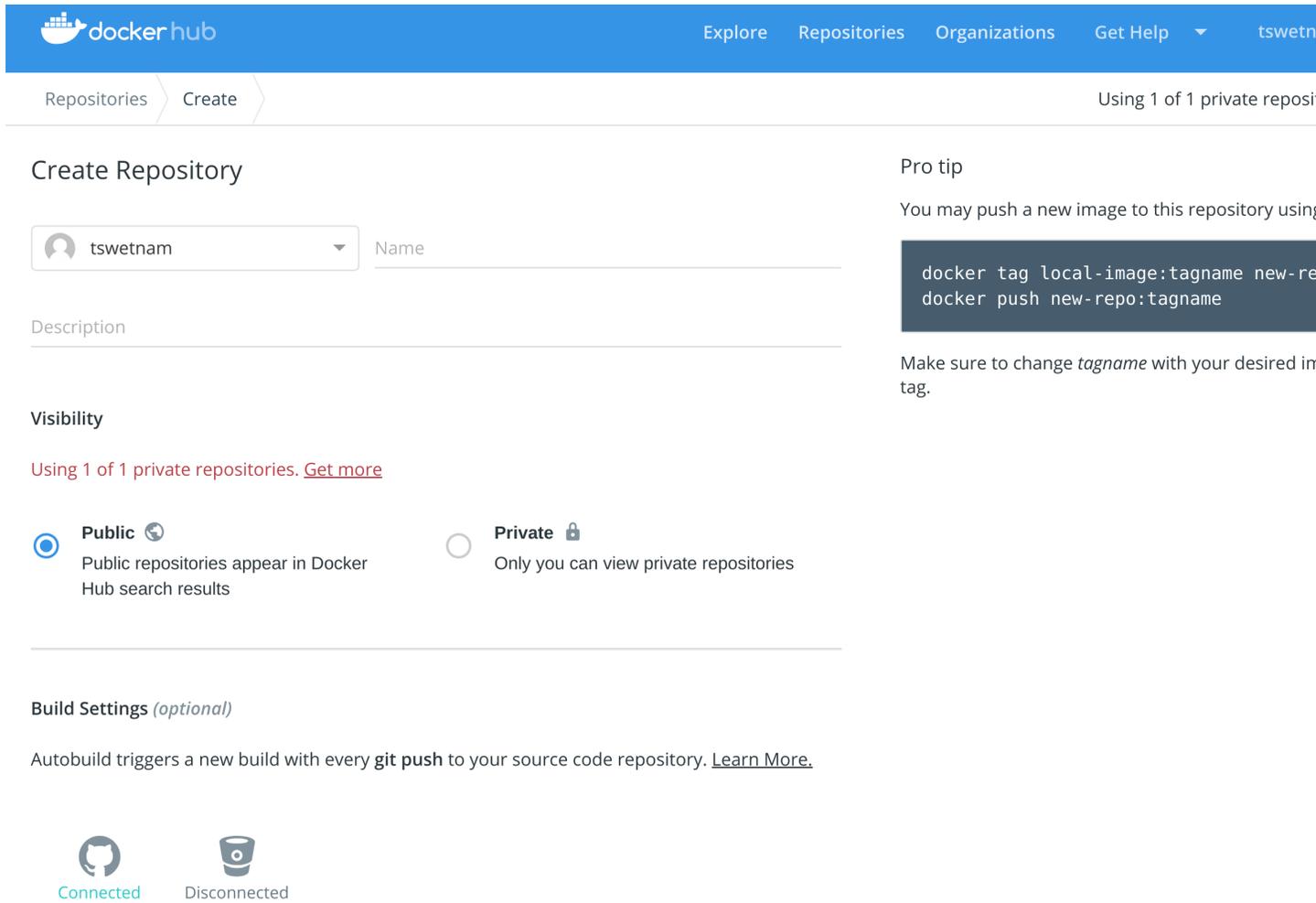
- If you have previously linked your Github or Bitbucket account, you must have chosen the Public and Private connection type. To view your current connection settings, log in to Docker Hub and choose Profile > Settings > Linked Accounts & Services.
- Building Windows containers is not supported.

2.2 Link your Docker Hub account to GitHub

1. Log into Docker Hub.
2. Click “Create Repository+”



3. Click the Build Settings and select GitHub.



Repositories > Create > Using 1 of 1 private repositories

Create Repository

 tswetnam

Description

Visibility

Using 1 of 1 private repositories. [Get more](#)

Public  Public repositories appear in Docker Hub search results

Private  Only you can view private repositories

Build Settings (optional)

Autobuild triggers a new build with every `git push` to your source code repository. [Learn More.](#)

 **Connected**  **Disconnected**

Pro tip

You may push a new image to this repository using

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

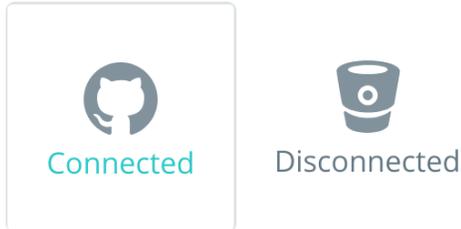
Make sure to change `tagname` with your desired image tag.

The system prompts you to choose between **Public and Private** and **Limited Access**. The **Public and Private** connection type is required if you want to use the Automated Builds.

4. Press `Select` under **Public and Private** connection type. If you are not logged into GitHub, the system prompts you to enter GitHub credentials before prompting you to grant access. After you grant access to your code repository, the system returns you to Docker Hub and the link is complete.

Build Settings *(optional)*

Autobuild triggers a new build with every **git push** to your source code repository. [Learn More](#)




 × ▼

▼ [Click here to customize the build settings](#)

BUILD RULES [+](#)

The build rules below specify how to build your source into Docker images.

Source Type	Source	Docker Tag	Dockerfile location	Build Caching
Branch ▼	master	latest	Dockerfile	<input checked="" type="checkbox"/>

▶ [View example build rules](#)

After you grant access to your code repository, the system returns you to Docker Hub and the link is complete. For example, github linked hosted repository looks like this:

The screenshot shows the Docker Hub interface for the repository `tswetnam/emsi-rstudio`. The 'Builds' tab is selected, showing a 'Build Activity' chart and 'Automated Builds' section.

Build Activity
 Overview of your build activity of the last 19 builds. The chart shows build duration in minutes (0 to 270) for Queue (light blue), Success (teal), Failed (red), and Cancelled (grey) builds.

Automated Builds
 Autobuild triggers a new build with every **git push** to your source code repository. [Learn More.](#)

`tyson-swetnam/emsi` | Use Docker Hub's infrastructure | Autotests: Internal and External Pull Requests

Docker Tag	Source	Build Status	Autobuild	Build caching	Triggers
3.5.2	master	SUCCESS	✓	✓	Trig

2.3 Automated Container Builds

Automated build repositories rely on the integration with a version control system (GitHub or Gitlab) where your Dockerfile is kept.

Let's create an automatic build for our container using the instructions below:

1. Initialize git repository for the `jupyter` directory you created for your Dockerfile

```
$ git init
Initialized empty Git repository in /home/username/github/repository_name/
$ git status
```

(continues on next page)

(continued from previous page)

```

On branch master

Initial commit

Untracked files:
(use "git add <file>..." to include in what will be committed)

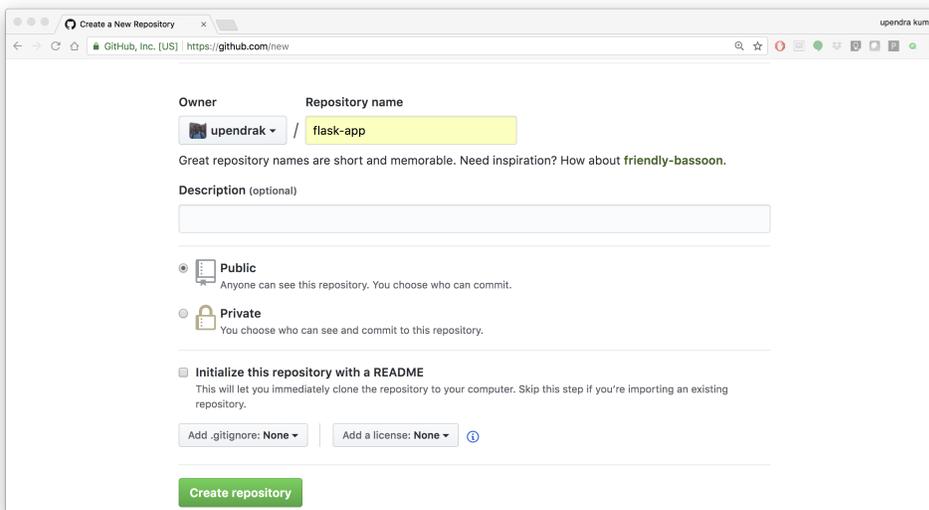
    Dockerfile

nothing added to commit but untracked files present (use "git add" to track)

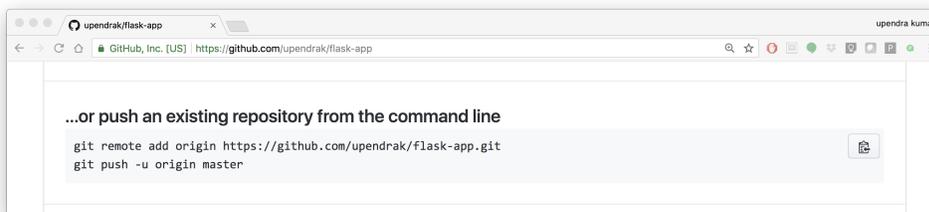
$ git add * && git commit -m"Add files and folders"
[master (root-commit) cdfd021] Add files and folders
 4 files changed, 75 insertions(+)
 create mode 100644 Dockerfile

```

2. Create a new repository on github by navigating to this url - <https://github.com/new>



3. Push the repository to github



```

$ git remote add origin https://github.com/username/jupyter.git

$ git push -u origin master
Counting objects: 7, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 1.44 KiB | 0 bytes/s, done.

```

(continues on next page)

(continued from previous page)

```
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/username/jupyter.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

4. Select `Create > Create Automated Build from Docker Hub`.

- The system prompts you with a list of User/Organizations and code repositories.
- For now select your GitHub account from the User/Organizations list on the left. The list of repositories change.
- Pick the project to build. In this case `jupyter`. Type in “Container Camp Jupyter” in the Short Description box.
- If you have a long list of repos, use the filter box above the list to restrict the list. After you select the project, the system displays the Create Automated Build dialog.

Build configurations

SOURCE REPOSITORY

 tyson-swetnam x emsi

NOTE: Changing source repository may affect existing build rules.

BUILD LOCATION

Build on Docker Hub's infrastructure

AUTOTEST

- Off
- Internal Pull Requests
- Internal and External Pull Requests

REPOSITORY LINKS

- Off
- Enable for Base Image 

BUILD RULES +

The build rules below specify how to build your source into Docker images.

Source Type	Source	Docker Tag	Dockerfile location	Build Context 	Autobuild	Build Caching
Branch	master	3.5.2	Dockerfile	/docker/rstud	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

▶ View example build rules

BUILD ENVIRONMENT VARIABLES +

Delete

Cancel

Save

Save and

Build triggers

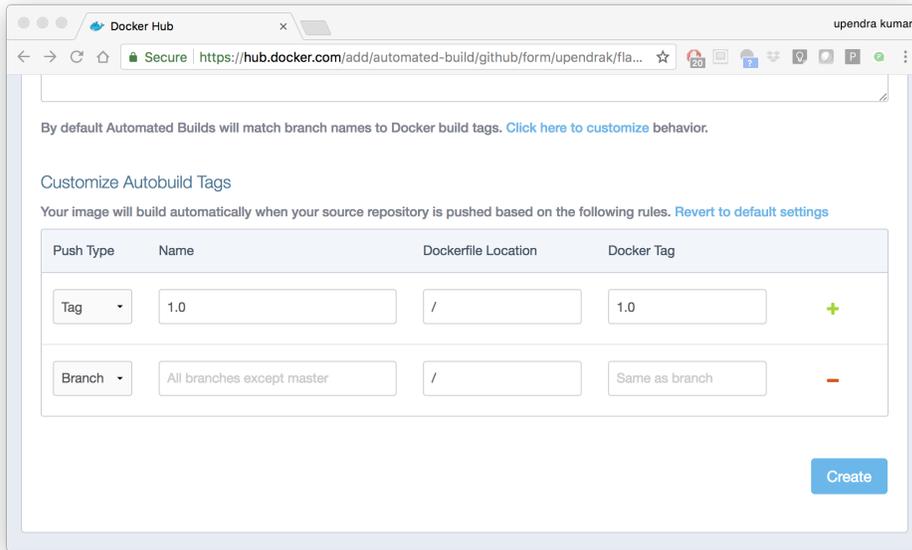
Trigger your Automated Build by sending a POST to a specific endpoint.

Trigger name +

Name Trigger Url

Note: The dialog assumes some defaults which you can customize. By default, Docker builds images for each branch in your repository. It assumes the Dockerfile lives at the root of your source. When it builds an image, Docker tags it with the branch name.

5. Customize the automated build by pressing the [Click here to customize behavior](#) link.



Specify which code branches or tags to build from. You can build by a code branch or by an image tag. You can enter a specific value or use a regex to select multiple values. To see examples of regex, press the [Show More](#) link on the right of the page.

- Enter the `master` (default) for the name of the branch.
- Leave the Dockerfile location as is.
- Recall the file is in the root of your code repository.
- Specify `1.0` for the Tag Name.

6. Click [Create](#).

Important: During the build process, Docker copies the contents of your Dockerfile to Docker Hub. The Docker community (for public repositories) or approved team members/orgs (for private repositories) can then view the Dockerfile on your repository page.

The build process looks for a `README.md` in the same directory as your Dockerfile. If you have a `README.md` file in your repository, it is used in the repository as the full description. If you change the full description after a build, it's overwritten the next time the Automated Build runs. To make changes, modify the `README.md` in your Git repository.

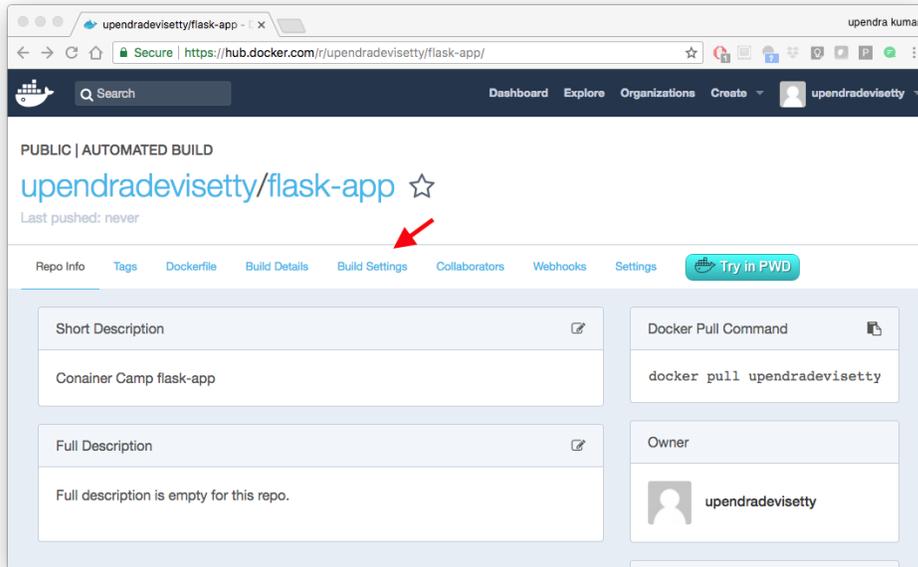
Warning: You can only trigger one build at a time and no more than one every five minutes. If you already have a build pending, or if you recently submitted a build request, Docker ignores new requests.

It can take a few minutes for your automated build job to be created. When the system is finished, it places you in the detail page for your Automated Build repository.

7. Manually Trigger a Build

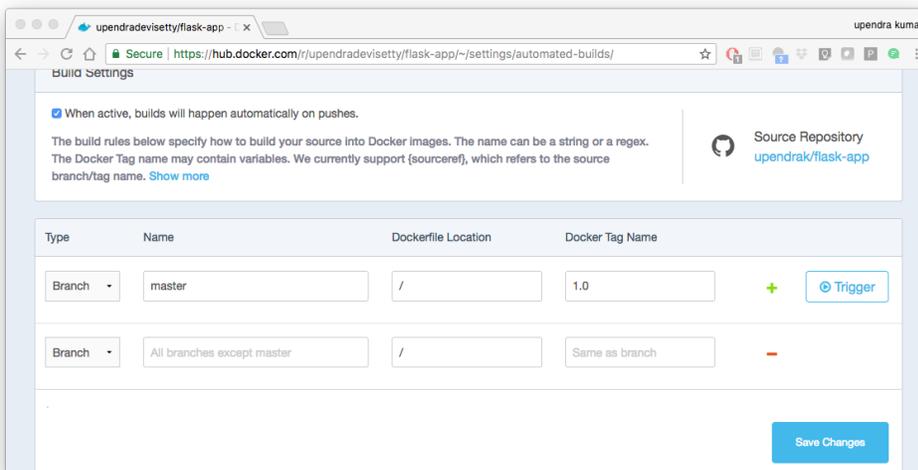
Before you trigger an automated build by pushing to your GitHub `jupyter` repo, you'll trigger a manual build. Triggering a manual build ensures everything is working correctly.

From your automated build page choose `Build Settings`



Press `Trigger` button and finally click `Save Changes`.

Note: Docker builds everything listed whenever a push is made to the code repository. If you specify a particular branch or tag, you can manually build that image by pressing the `Trigger`. If you use a regular expression syntax (regex) to define your build branch or tag, Docker does not give you the option to manually build.



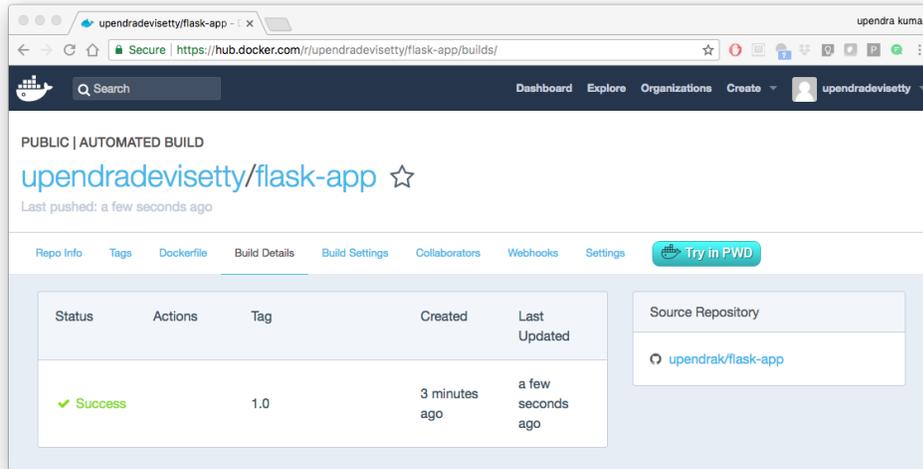
8. Review the build results

The Build Details page shows a log of your build systems:

Navigate to the `Build Details` page.

Wait until your image build is done.

You may have to manually refresh the page and your build may take several minutes to complete.



Exercise 1 (5-10 mins): Updating and automated building

- `git add, commit` and push to your GitHub or Gitlab repo
- Trigger automatic build with a new tag (2.0) on Docker Hub
- Pull your Docker image from Docker Hub to a new location.
- Run the instance to make sure it works

1.34.3 3. Managing Data in Docker

It is possible to store data within the writable layer of a container, but there are some limitations:

- The data doesn't persist when that container is no longer running, and it can be difficult to get the data out of the container if another process needs it.
- A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.
- Its better to put your data into the container **AFTER** it is build - this keeps the container size smaller and easier to move across networks.

Docker offers three different ways to mount data into a container from the Docker host:

- **volumes,**
- **bind mounts,**
- **tmpfs volumes.**

When in doubt, volumes are almost always the right choice.

3.1 Volumes

Volumes are created and managed by Docker. You can create a new volume explicitly using the `docker volume create` command, or Docker can create a volume in the container when the container is built.

When you run a container, you can bring a directory from the host system into the container, and give it a new name and location using the `-v` or `--volume` flag.

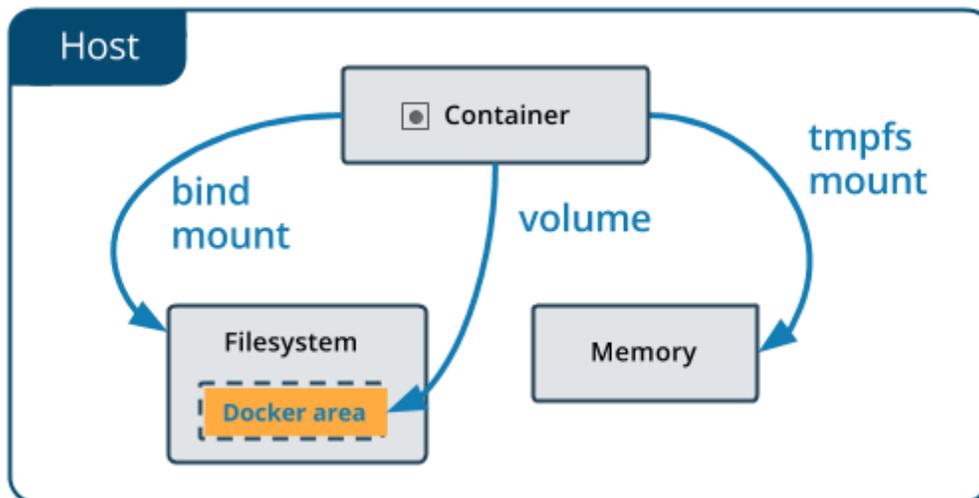
```
$ docker run -v /home/username/your_data_folder:/data username/jupyter:latest
```

In the example above, you can mount a folder from your localhost, in your home user directory into the container as a new directory named `/data`.

When you create a Docker volume, it is stored within a directory on the Docker Linux host (`/var/lib/docker/`

Note: File location on Mac OS X is a bit different. [see here](https://timonweb.com/posts/getting-path-and-accessing-persistent-volumes-in-docker-for-mac/) <<https://timonweb.com/posts/getting-path-and-accessing-persistent-volumes-in-docker-for-mac/>>‘_.

A given volume can be mounted into multiple containers simultaneously. When no running container is using a volume, the volume is still available to Docker and is not removed automatically. You can remove unused volumes using `docker volume prune` command.



Volumes are often a better choice than persisting data in a container’s writable layer, because using a volume does not increase the size of containers using it, and the volume’s contents exist outside the lifecycle of a given container. While bind mounts (which we will see later) are dependent on the directory structure of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- A new volume’s contents can be pre-populated by a container.

Note: If your container generates non-persistent state data, consider using a `tmpfs` mount to avoid storing the data anywhere permanently, and to increase the container’s performance by avoiding writing into the container’s writable

layer.

3.1.1 Choose the `-v` or `--mount` flag for mounting volumes

Originally, the `-v` or `--volume` flag was used for standalone containers and the `--mount` flag was used for swarm services. However, starting with Docker 17.06, you can also use `--mount` with standalone containers. In general, `--mount` is more explicit and verbose. The biggest difference is that the `-v` syntax combines all the options together in one field, while the `--mount` syntax separates them. Here is a comparison of the syntax for each flag.

Tip: New users should use the `--mount` syntax. Experienced users may be more familiar with the `-v` or `--volume` syntax, but are encouraged to use `--mount`, because research has shown it to be easier to use.

`-v` or `--volume`: Consists of three fields, separated by colon characters (:). The fields must be in the correct order, and the meaning of each field is not immediately obvious.

- In the case of named volumes, the first field is the name of the volume, and is unique on a given host machine.
- The second field is the path where the file or directory are mounted in the container.
- The third field is optional, and is a comma-separated list of options, such as `ro`.

3.2 Bind mounts

`--mount`: Consists of multiple key-value pairs, separated by commas and each consisting of a `<key>=<value>` tuple. The `--mount` syntax is more verbose than `-v` or `--volume`, but the order of the keys is not significant, and the value of the flag is easier to understand. - The type of the mount, which can be **bind**, **volume**, or **tmpfs**. - The source of the mount. For named volumes, this is the name of the volume. For anonymous volumes, this field is omitted. May be specified as **source** or **src**. - The destination takes as its value the path where the file or directory is mounted in the container. May be specified as **destination**, **dst**, or **target**. - The readonly option, if present, causes the bind mount to be mounted into the container as read-only.

Note: The `--mount` and `-v` examples have the same end result.

3.3 Create and manage volumes

Unlike a bind mount, you can create and manage volumes outside the scope of any container.

Let's create a volume

```
$ docker volume create my-vol
```

List volumes:

```
$ docker volume ls
local                my-vol
```

Inspect a volume by looking at the Mount section in the `docker volume inspect`

```
$ docker volume inspect my-vol
[
  {
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
    "Name": "my-vol",
    "Options": {},
    "Scope": "local"
  }
]
```

Remove a volume

```
$ docker volume rm my-vol
```

3.3.1 Populate a volume using a container

This example starts an nginx container and populates the new volume `nginx-vol` with the contents of the container's `/var/log/nginx` directory, which is where Nginx stores its log files.

```
$ docker run -d -p 8889:80 --name=jupytertertest --mount source=jupyter-vol,target=/var/
↳log/jupyter username/jupyter:latest
```

So, we now have a copy of Jupyter volume running inside a Docker container on our machine, and our host machine's port 5000 maps directly to that copy of Jupyter's port 80. Let's use curl to do a quick test request:

```
cat jupyter-vol/_data/access.log
```

Use `docker inspect jupyter-vol` to verify that the volume was created and mounted correctly. Look for the Mounts section:

```
"Mounts": [
  {
    "Type": "volume",
    "Name": "jupyter-vol",
    "Source": "/var/lib/docker/volumes/jupyter-vol/_data",
    "Destination": "/var/log/jupyter",
    "Driver": "local",
    "Mode": "z",
    "RW": true,
    "Propagation": ""
  }
],
```

This shows that the mount is a volume, it shows the correct source and destination, and that the mount is read-write.

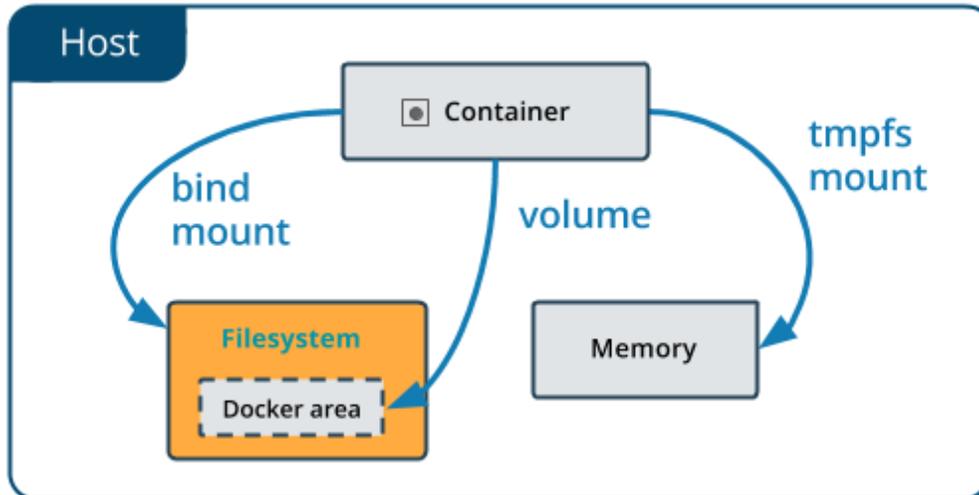
After running either of these examples, run the following commands to clean up the containers and volumes.

```
$ docker stop jupytertertest
$ docker rm jupytertertest
$ docker volume rm jupyter-vol
```

3.4 Bind mounts

Bind mounts: When you use a bind mount, a file or directory on the host machine is mounted into a container.

Tip: If you are developing new Docker applications, consider using named **volumes** instead. You can't use Docker CLI commands to directly manage bind mounts.



Warning: One side effect of using bind mounts, for better or for worse, is that you can change the host filesystem via processes running in a container, including creating, modifying, or deleting important system files or directories. This is a powerful ability which can have security implications, including impacting non-Docker processes on the host system.

If you use `--mount` to bind-mount a file or directory that does not yet exist on the Docker host, Docker does not automatically create it for you, but generates an error.

3.2.1 Start a container with a bind mount

```
$ mkdir data
$ docker run -p 8888:8888 --name jupytertertest --mount type=bind,source="$(pwd)"/data,
↪target=/var/log/jupytertertest username/jupyter:latest
```

Use `docker inspect jupytertertest` to verify that the bind mount was created correctly. Look for the “Mounts” section

This shows that the mount is a bind mount, it shows the correct source and target, it shows that the mount is read-write, and that the propagation is set to rprivate.

Stop the container:

```
$ docker rm -f jupytertertest
```

3.4.1 Use a read-only bind mount

For some development applications, the container needs to write into the bind mount, so changes are propagated back to the Docker host. At other times, the container only needs read access.

This example modifies the one above but mounts the directory as a read-only bind mount, by adding `ro` to the (empty by default) list of options, after the mount point within the container. Where multiple options are present, separate them by commas.

```
$ docker run -d -p 8888:8888 --name jupytertertest --mount type=bind,source="$(pwd)"/
↳data,target=/var/log/jupyter,readonly username/jupyter:latest
```

Use `docker inspect jupytertertest` to verify that the bind mount was created correctly. Look for the Mounts section:

Stop the container:

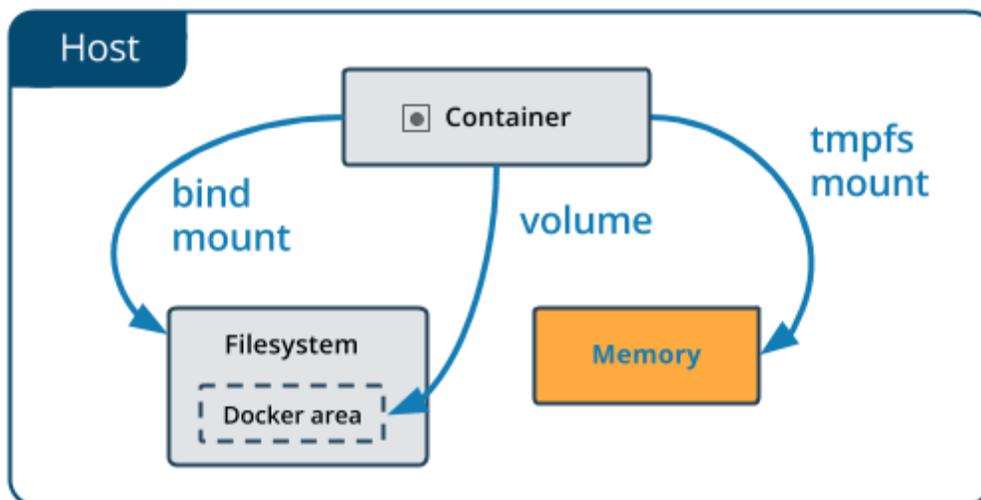
```
$ docker rm -f jupytertertest
```

Remove the volume:

```
$ docker volume rm jupytertertest
```

3.3 tmpfs Mounts

tmpfs mounts: A tmpfs mount is not persisted on disk, either on the Docker host or within a container. It can be used by a container during the lifetime of the container, to store non-persistent state or sensitive information. For instance, internally, swarm services use tmpfs mounts to mount secrets into a service's containers.



Volumes and **bind mounts** are mounted into the container's filesystem by default, and their contents are stored on the host machine. There may be cases where you do not want to store a container's data on the host machine, but you also don't want to write the data into the container's writable layer, for performance or security reasons, or if the data relates to non-persistent application state. An example might be a temporary one-time password that the container's application creates and uses as-needed. To give the container access to the data without writing it anywhere permanently, you can use a tmpfs mount, which is only stored in the host machine's memory (or swap, if memory is low). When the container stops, the tmpfs mount is removed. If a container is committed, the tmpfs mount is not saved.

```
$ docker run -d -p 8888:8888 --name jupytertertest --mount type=tmpfs,target=/var/log/
↳jupyterter username/jupyterter:latest
```

Use `docker inspect jupytertertest` to verify that the bind mount was created correctly. Look for the Mounts section:

```
$ docker inspect jupytertertest
```

You can see from the above output that the `Source` field is empty which indicates that the contents are not available on Docker host or host file system.

Stop the container:

```
$ docker rm -f jupytertertest
```

Remove the volume:

```
$ docker volume rm jupytertertest
```

1.34.4 4. Docker Compose for multi-container apps

Docker Compose is a tool for defining and running your multi-container Docker applications.

Main advantages of Docker compose include:

- Your applications can be defined in a YAML file where all the options that you used in `docker run` are now defined (Reproducibility).
- It allows you to manage your application as a single entity rather than dealing with individual containers (Simplicity).

Let's now create a simple web app with Docker Compose using Flask (which you already seen before) and Redis. We will end up with a Flask container and a Redis container all on one host.

Note: The code for the above compose example is available [here](#)

1. You'll need a directory for your project on your host machine:

```
$ mkdir compose_flask && cd compose_flask
```

2. Add the following to `requirements.txt` inside `compose_flask` directory:

```
flask
redis
```

3. Copy and paste the following code into a new file called `app.py` inside `compose_flask` directory:

```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    redis.incr('hits')
```

(continues on next page)

(continued from previous page)

```

return 'This Compose/Flask demo has been viewed %s time(s).' % redis.get('hits')

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)

```

4. Create a Dockerfile with the following code inside `compose_flask` directory:

```

FROM python:2.7
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD python app.py

```

5. Add the following code to a new file, `docker-compose.yml`, in your project directory:

```

version: '2'
services:
  web:
    restart: always
    build: .
    ports:
      - "8888:5000"
    volumes:
      - ./code
    depends_on:
      - redis
  redis:
    restart: always
    image: redis

```

A brief explanation of `docker-compose.yml` is as below:

- `restart: always` means that it will restart whenever it fails.
- We define two services, **web** and **redis**.
- The web service builds from the Dockerfile in the current directory.
- Forwards the container's exposed port (5000) to port 8888 on the host.
- Mounts the project directory on the host to `/code` inside the container (allowing you to modify the code without having to rebuild the image).
- `depends_on` links the web service to the Redis service.
- The redis service uses the latest Redis image from Docker Hub.

Note: Docker for Mac and Docker Toolbox already include Compose along with other Docker apps, so Mac users do not need to install Compose separately. Docker for Windows and Docker Toolbox already include Compose along with other Docker apps, so most Windows users do not need to install Compose separately.

For Linux users

```

sudo curl -L https://github.com/docker/compose/releases/download/1.19.0/docker-
↪compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

```

5. Build and Run with `docker-compose up -d` command

```

$ docker-compose up -d

Building web
Step 1/5 : FROM python:2.7
2.7: Pulling from library/python
f49cf87b52c1: Already exists
7b491c575b06: Already exists
b313b08bab3b: Already exists
51d6678c3f0e: Already exists
09f35bd58db2: Already exists
f7e0c30e74c6: Pull complete
c308c099d654: Pull complete
339478b61728: Pull complete
Digest: sha256:8cb593cb9cd1834429f0b4953a25617a8457e2c79b3e111c0f70bffd21acc467
Status: Downloaded newer image for python:2.7
---> 9e92c8430ba0
Step 2/5 : ADD . /code
---> 746bcecf3c9
Step 3/5 : WORKDIR /code
---> c4cf3d6cb147
Removing intermediate container 84d850371a36
Step 4/5 : RUN pip install -r requirements.txt
---> Running in d74c2e1c7bf7
Collecting flask (from -r requirements.txt (line 1))
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting redis (from -r requirements.txt (line 2))
  Downloading redis-2.10.6-py2.py3-none-any.whl (64kB)
Collecting itsdangerous>=0.21 (from flask->-r requirements.txt (line 1))
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting Jinja2>=2.4 (from flask->-r requirements.txt (line 1))
  Downloading Jinja2-2.10-py2.py3-none-any.whl (126kB)
Collecting Werkzeug>=0.7 (from flask->-r requirements.txt (line 1))
  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
Collecting click>=2.0 (from flask->-r requirements.txt (line 1))
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->flask->-r requirements.txt (line 1))
  Downloading MarkupSafe-1.0.tar.gz
Building wheels for collected packages: itsdangerous, MarkupSafe
  Running setup.py bdist_wheel for itsdangerous: started
  Running setup.py bdist_wheel for itsdangerous: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/fc/a8/66/
  ↪24d655233c757e178d45dea2de22a04c6d92766abfb741129a
  Running setup.py bdist_wheel for MarkupSafe: started
  Running setup.py bdist_wheel for MarkupSafe: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/88/a7/30/
  ↪e39a54a87bcbe25308fa3ca64e8ddc75d9b3e5afa21ee32d57
Successfully built itsdangerous MarkupSafe
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, Werkzeug, click, ↪
  ↪flask, redis
Successfully installed Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7 flask-0.
  ↪12.2 itsdangerous-0.24 redis-2.10.6
---> 5cc574ff32ed
Removing intermediate container d74c2e1c7bf7
Step 5/5 : CMD python app.py
---> Running in 3ddb7040e8be
---> e911b8e8979f

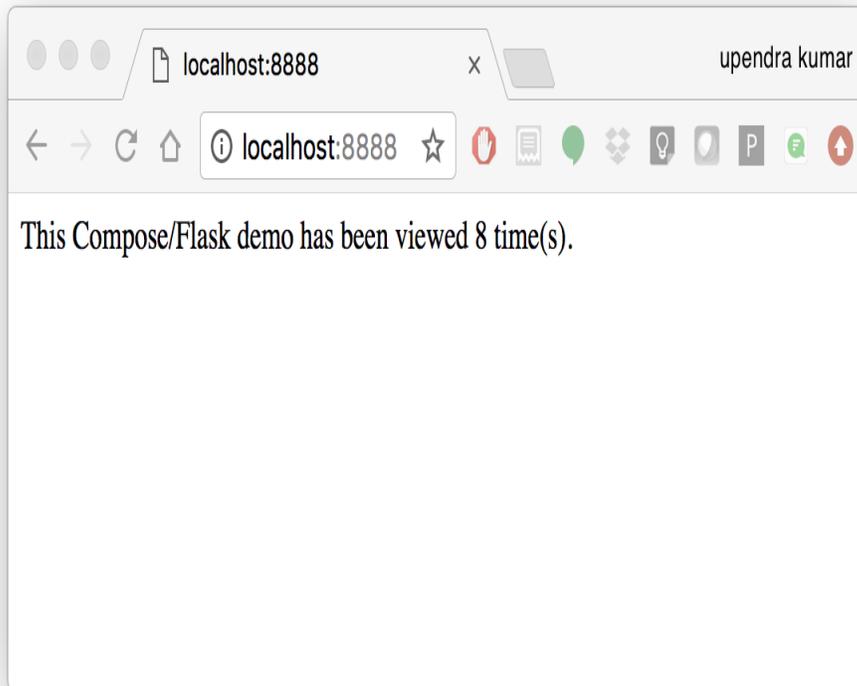
```

(continues on next page)

(continued from previous page)

```
Removing intermediate container 3ddb7040e8be
Successfully built e911b8e8979f
Successfully tagged composeflask_web:latest
```

And that's it! You should be able to see the Flask application running on `http://localhost:8888` or `<ipaddress>:8888`



```
$ cat output.txt
Prediction of DecisionTreeClassifier:['apple' 'orange' 'apple']
```



 Learning Center Home

1.35 Introduction to R & RStudio

1.35.1 Setup

1. You need to download R & RStudio:
 - [Download R](#)
 - [Download R Studio](#)
2. Move to the Applications folder.
3. Open RStudio.

Go to Session -> Set Working Directory to set where you will pull data files from and/or save your code.

1.35.2 Introduction

We will learn how to: - navigate & interact with R Studio

- UI of R Studio
- how to use “help”
- install packages
- upload data
- **data structures**
 - strings, factors, numbers, integers
 - vectors & arrays
 - matrices & lists
- **explore data**
 - data manipulation
 - data subsetting

R Studio makes using R programming language easier to interact with and to keep track of projects.

1.35.3 Navigating & Interacting with R Studio

Basic Layout

IR Console

The basic layout includes: - Interactive R Console (left) <- most of your time will be spent here - Environment/History (upper right) - Files/Plots/Packages/Help/Viewer (lower right)

Once you open a new R script (File -> New File -> R Script), and editor panel should appear in the upper left. R scripts are saved as .R files. These can be rearranged by going into Preferences.

Calculating with R

Note spaces don't matter unless it's in the middle of an argument

Note R starts counting at 1, not at 0

Using R as a calculator

```
> 2+2
> 4

> 1 +
> + #R will let you know that the code is incomplete

> 2/10000
> 2e-04
```

Exercise:

1. What is the output for 5e3?
2. How would you add 5 and 3 and multiply the sum by 2?

Comparing things: Using logical operators

```
> 1 == 1 #spaces between logical arguments matter
> TRUE

> 1 < 2
> TRUE

> 1 >= 9
> FALSE
```

Other logical conditions: &, |, !

HELP!

help() is the most useful function in R. You will likely use this and Stack Overflow to help solve most of your problems (not life problems, you're on your own for that).

```
help(plot)
```

Parts of the help file: - Description

This describes what the function does.

- **Usage** This describes the formula and arguments for the function
- **Arguments** These are different inputs into the function that can be used. The argument (e.g., x, y) do not always need to be specified. For example,

```
plot(x = data.x, y = data.y)
plot(data.x, data.y)
```

are the same thing.

- **Details** Usually these state the outputs of the function, or any other nuance within the function that may not be obvious.
- **See also** This will link to similar functions, or functions that can be called with this function.

- **Examples** Some are better than others. Generally, though, this gives examples of the arguments most commonly used in the function.

Searching for help: - type in error message (just delete words specific to your data) - include in package name - type “CRAN” after to help search for R programming specifically

Installing packages

```
install.packages("packageName")
```

To install more than one package at once you can use the `c("package1", "package2")` concatenate:

```
install.packages(c("package1", "package2"))
```

Often installing a package will install its dependencies as well. You can set the dependency installation by hand using:

```
install.packages("packageName", dependencies=TRUE)
install.packages(c("package1", "package2"), dependencies=TRUE)
```

You can see installed packages with the following command:

```
installed.packages()
```

To use the package after it’s been loaded:

Uploading Data

There are many ways to upload data in the R environment depending on the document type you have.

```
#General reading
read.table("dataFile.txt", sep = "/t")
```

Exercise:

1. What are the arguments for `read.table()`?
2. What arguments would you use to upload a .csv file using `read.table()`?

```
#.csv files
read.csv()

#reading in from an online source or path to the directory if you're not in the right_
↳working directory
read.table(path/to/file)
```

1.35.4 Data Structures

Types of Variables

Character - text that cannot have calculations done on them e.g., “a”, “xyz”

Numeric - numerical values include decimals and can have calculations performed on them e.g., 1, 1.5

Integer - whole numbers only, and can also have calculations performed on them e.g., 2L (L stores it as an integer)

Logical - TRUE or FALSE

Exercise:

1. What does the following return? What does it mean?

```
str(10)
str("10")
```

Try calculations on the following.

2. What works and what doesn't? Why or why not?

```
10*2
"10"*2
```

Errors v. Warnings: Errors are given when R cannot perform the calculation Warnings mean that the function has run but perhaps with some issues.

Storing Variables

We can assign any of the types of data above in a “place holder”. Variables are assignee using “<-“.

For example, we can store the number 10 in a letter to use later

```
a <- 10
```

NOTE Do not create variables that are already functions or arguments (e.g., c, T, F). **NOTE** Do not overwrite variables.

Exercise:

1. What does a*2 give you?

Vectors

Vectors are 1-D object that contain “like” data types. You can create a string of variables and add to a vector using c(), which is short for concatenate.

Exercise:

1. What are the outputs of the code below?
2. Create your own vector using the vector() function.

```
x <- c(1, 2, 3, 4, 5)
y <- 1:5
z <- seq(1, 5, 1)
```

3. Are x, y, and z all the same structure? If not, how would you make them all the same?

Adding to vectors: the concatenate function: c()

```
d <- 1
d <- c(d, 2)
```

Try adding two to every number in the vector “x”.

3. How do you add two to every number in x?

What happens when you add a character to a vector?

ATOMIC VECTORS are vectors which cannot be simplified anymore, and therefore “\$” cannot be used on them. Yes, this error happens a lot. Yes, it is frustrating. Good luck.

Matrices & Dataframes

A matrix and a dataframe are both 2-D objects that are made up of vectors.

Creating a dataframe using `data.frame()`

Exercise:

1. Play with the different types of data in the `data.frame()`. What happens?

You can combine dataframes:

```
hello <- data.frame(1:26, letters, words = c("hey", "you"))
hi <- data.frame(1:26, letters, c("hey", "you"))
howdy <- data.frame(hello, hi)
```

How do you name the column with the numbers 1-26?

What are the column headers? What happens when you do the following?

Adding columns and rows using `cbind()` and `rbind()`

```
cbind(hello, "goodbye")
```

We can call columns using `$` in the form of `data.frame$column` or call them using the modifier `data.frame[row#, column#]`

Calling columns:

```
hello[,2] #[] are like an index
hello$letters
```

Subsetting:

Useful Functions to explore data types

```
View() #can also double click on dataframe inside the R environment tab
str()
summary()
class()
typeof()
length()
attributes() #can also click on dataframe inside the R environment tab
dim()
head()
tail()
```

Exercise

1. What is the output?

```
hello[,-2]
```

Likewise, columns and rows can be removed using “-” as a modifier

You can save a dataframe using `write.table()` and `write.csv()`.

NOTE do not overwrite your dataset!!

If you rerun a script, you may overwrite your results or new data. Put a “#” after use!

The R Environment

You can view your environment either by looking at the upper left tab or by typing the following:

```
ls() #see variables in your environment
```

You can remove objects using the rm() function.

Exercise:

1. How would you remove “a” from the environment? How would you check?

1.35.5 Exploring Data

Data Manipulation

Create the following dataframe:

```
cats <- data.frame(coat = c("calico", "black", "tabby"),
                  weight = c(2.1, 5.0, 3.2),
                  likes_string = c(1, 0, 1))
class(cats)
```

Let’s add!

```
cats$weight + 2
cats$coat + cats$coat
```

What are the outputs?

We can use the function “paste” to make more complex strings:

```
paste("My cat is", cats$coat)
```

What is the output?

Subsetting Data

Exercise:

1. What is the function for subsetting data?
2. What are the outputs?

```
x <- c(a=5.4, b=6.2, c=7.1, d=4.8, e=7.5) # we can name a vector 'on the fly'
#x is a vector
x[c(a,c),]
x[names(x) == "a"]
x[names(x) == "a" | "c"]
x[names(x) != "a"]
```

1.35.6 Terminal

Can run terminal in RStudio. This is useful if you want to run a program and still be able to use R, or if you need dependencies. Also, the terminal does not interact with the R environment.

Tools -> Terminal -> New Terminal

- Send feedback: Tutorials@CyVerse.org



1.36 RStudio with Version Control

RStudio can be used with Git or SVN. For today's lesson we're going to be using Git with your new GitHub accounts.

Some things to remember about the platform

- You can pull other GitHub repositories using `git clone` via the RStudio terminal.
- Your local commits are not sent to GitHub until you initiate the repository with your GitHub username and password.

1.36.1 Instructions

1. Create a GitHub Repository
2. On your local computer, create a directory called `~/github` and change into that directory
3. Open RStudio and start a new Project: *File > New Project > Version Control > Git*. In the repository URL paste the URL of your new GitHub repository
4. Save your new git repository to the newly created `~/github` directory.
5. Open the directory in RStudio and set the project folder as the workspace.

1.36.2 R Markdown

1. Create a new `.Rmd` format file.
2. Set the parameters (e.g. HTML, PDF, etc)
3. Create an code block and specify the language, e.g.

1.36.3 Workflow R

R has many projects which deal with workflows

We're going to talk about the `workflowr` and `drake`

1. Install Packages and depends
2. Follow instructions for building a `workflowr` website
3. Follow instructions for building a `drake` workflow

Self Paced

rOpenSci is a leader in the development of reproducible research. There are hundreds of repositories to explore for examples of research using R.

[|CyverseLogol_|LearningCenter|_](#)

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-



Learning Center Home

1.37 GitHub Action and EHT Demo

1.37.1 Introduction

Building Docker images on your machine and then pushing it to DockerHub manually is sometime cumbersome. Since we use Git and GitHub to keep track of all our source codes anyway, it will be great to ask GitHub to automatically build and push docker images for us whenever we change our codes.

We will use one of the Event Horizon Telescope (EHT)'s imaging pipeline as an example. The EHT is an international collaboration with a science goal to take pictures of black holes. The EHT published its visibility data on [CyVerse](#) and its software pipeline on [GitHub](#).

1.37.2 Fork the EHT Pipeline

When you are on the EHT pipeline GitHub repository, first, click the “Fork” badge/button to create your own repository.

In your own repository, go to “Settings” to rename your repository to something readable for human, e.g., “eht-demo”.

Because we will need to connect to DockerHub to publish our image, go to the “Secerts” tab in “Settings” and add the following two secrets:

```
DOCKERHUB_USERNAME : your Docker Hub username
DOCKERHUB_PASSWORD : your Docker Hub password
```

1.37.3 Create a Dockerfile

To edit your version of the pipeline repository, let’s clone it to your “local” machines (laptop, desktop, atmosphere VM, etc).

```
git clone git@github.com:[GITHUB_USERNAME]/eht-demo.git
```

Then, change-directory into your local repository and create a `wrapper.sh` script:

```
#!/bin/bash
python /usr/local/src/eht-imaging_pipeline.py "$@"
```

Turn it into an run-able script by

```
chmod 755 wrapper.sh
```

Then, add a new Dockerfile with the follow content:

```
FROM eventhorizontelescope/img-env
COPY eht-imaging/eht-imaging_pipeline.py /usr/local/src/
COPY wrapper.sh /usr/bin
WORKDIR /img
ENTRYPOINT ["/usr/bin/wrapper.sh"]
```

Commit all your files and push to GitHub:

```
git add .
git commit -m 'For building FOSS demo Docker image'
git push
```

1.37.4 Setup GitHub Action

Go back to your web browser and make sure that your own pipeline repository is updated. Click the “Actions” tab. Because you have not set up any GitHub Action, GitHub presents you many examples. Let’s click on “Workflow for Python, Maven, Docker and more . . .” at the bottom of the page and look for the “Docker image” example. Click “Set up this workflow” as a starting point.

GitHub now presents you an online text editor that describes an Action Workflow. Let’s just click “Start commit” to turn this into a Git commit.

Once you are done, click the “Actions” tab again. You will see the workflow is now set up. It’s probably still in the GitHub Action work queue. Wait a bit and it will turn into a running state.

Your “Action” workflow should finish successfully. However, it built an image call `my-image-name` inside the Action build machine. The name is not right, and you cannot use this Docker image.

1.37.5 Edit GitHub Action

Click on the name your Action workflow and select the “Workflow file” tab, then click the pencil icon on the top right, GitHub gives you an online editor again. Update the `dockerimage.yml` file to:

```
name: Docker Image CI

on: [push]

jobs:

  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Build the Docker image
      run: docker build . --file Dockerfile --tag ${ secrets.DOCKERHUB_USERNAME }/
↪eht-demo
    - name: Login to Docker Hub
      run: echo ${ secrets.DOCKERHUB_PASSWORD } | docker login -u ${ secrets.
↪DOCKERHUB_USERNAME } --password-stdin
    - name: Push to Docker Hub
      run: docker push ${ secrets.DOCKERHUB_USERNAME }/eht-demo:latest
```

Once you are done, commit it. And go back to “Actions” tab. Because editing the Action workflow is itself a Git commit, it triggers GitHub Action to rerun the workflow. If it works, you it should have built a Docker image and push it to Docker Hub.

1.37.6 EHT Image Reconstruction

Now, we are ready to perform an EHT image reconstruction to create your own black hole image!

First, because it will take some time to download the Docker image, let’s start pulling it first onto your “local” machines (laptop, desktop, atmosphere VM, etc) in the background.

```
docker pull [DOCKERHUB_USERNAME]/eht-demo > git-pull-log &
```

Let’s also create an empty work directory

```
mkdir ~/eht-demo
cd ~/eht-demo
```

Remember EHT published its data on CyVerse? Let’s download a data file:

```
wget https://de.cyverse.org/anon-files//iplant/home/shared/commons_repo/curated/EHTC_
↪FirstM87Results_Apr2019/uvfits/SR1_M87_2017_095_lo_hops_netcal_StokesI.uvfits
```

You have both the data and software (in a Docker image). Let’s perform the image reconstruction:

```
docker run --rm -v $PWD:/img [DOCKERHUB_USERNAME]/eht-demo -i SR1_M87_2017_095_lo_
↳hops_netcal_StokesI.uvfits -o [NAME].fits --savepdf
```

It will take some time. And it may work or may not work—depending on if you pull an old version of the pipeline repository, or a new version.

If docker finished with an error message, you used the old version. And it is a great opportunity to see how convenient to see how GitHub Actions work. If docker finished without an error, congratulations, you can skip the next paragraph.

Go to GitHub on your browser, open `eht-imaging/eht-imaging_pipeline.py`, and click the pencil icon to edit the file directly. Add the following two lines:

```
import matplotlib
matplotlib.use('Agg')
```

on line 50 and the commit from the GitHub website. Because GitHub Actions “license” for changes on your repository, it will automatically rebuild your Docker image. Once it is done, pull the new Docker image by:

```
docker pull [DOCKERHUB_USERNAME]/eht-demo
```

And now you can rerun your analysis with the new Docker image!

Once it’s done, you will see two new files `[NAME].fits` and `[NAME].pdf` on the local machines.

Here you go! You just reconstructed your own black hole image!

1.37.7 Exercise

OK this is cool so far. But the point of GitHub Actions is that it will rerun the workflow whenever you commit and push your repository to GitHub. So try to make some silly changes to your local Git repository, push it to GitHub, and see Actions react to your edits.



 Learning Center Home

1.38 GitHub Pages - a quick start

This is a quick introduction to , a simple way to use to set up a small website written in . This page won’t do everything, but you can throw up a basic website, use , and extend it.

1. Go to . Login, or if you don’t have an account get one and login.
2. Go to the “+” icon on the upper right and select **New repository**.
3. Enter a name for your repository (e.g. “profile”). Enter a description, and leave the repository as public. Select “**Initialize this repository with a README**”. If desired select a license. Finally click **Create repository**.

4. Look for the **Settings** menu (upper right, next to a “gear” icon). Scroll down to **GitHub Pages** and choose **master** branch and save your selection. Then Choose a theme and select your theme. You will be asked to **Commit changes**.
5. Your website will be visible at *https://GITHUBUSERNAME.github.io/REPONAME/*. (be sure to change *GITHUBUSERNAME* to your username, and *REPONAME* to the name you selected for your repo.)
6. You can edit your website by editing the readme file as desired.

Tip: You can preview how your looks using and editor like .

Fix or improve this documentation:

- On Github:
 - Send feedback: Tutorials@CyVerse.org
-

CyVerse Homepage: <http://www.cyverse.org>

CHAPTER 2

Funding and Citations

CyVerse is funded by the National Science Foundation under Award Numbers DBI-0735191, DBI-1265383, and DBI-1743442.

Please cite CyVerse appropriately when you make use of our resources, [CyVerse citation policy](#)

CHAPTER 3

License

Documentation contained in this repo is made available under CC BY 4.0 License: <https://creativecommons.org/licenses/by/4.0/legalcode>

Fix or improve this documentation

- Search for an answer:
 - Ask us for help: click  on the lower right-hand side of the page
 - Report an issue or submit a change: [Github Repo Link](#)
 - Send feedback: Tutorials@CyVerse.org
-