Git + Docker tutorial

Tony Wildish
Preamble

• This presentation, the tutorial material
  – https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip

• Pre-requisites
  – See https://bitbucket.org/TWildish/git-docker-tutorial/overview
  – Please tell me you did that already 😊

• Today:
  – 3:00 – 4:00: git overview + hands-on exercises
  – 4:00 – 5:00: docker overview + hands-on exercises
  – Familiarity with what’s possible, rather than a deep-dive
  – Worked examples of how to do things
This tutorial

• **Git**
  – Basics of repositories, local and remote
  – How to recover from mistakes
  – Working with branches
  – Working with teams

• **Docker**
  – Various ways to run & manage docker containers
  – A real bioinformatics application example
    • Thanks to Michael Barton
  – How to get data into/out of a docker container
  – How to build a simple docker container
  – Shifter – docker on Cori, Edison, and (eventually) Genepool

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Git history

• Git is a ‘Version Control System’, (VCS)
• Git manages collections of files (text, small binaries)
  – Tracks their history, versions
  – Tracks multiple development paths
  – Lets you recover previous versions
• Git is the VCS, don’t bother with anything else
  – CVS: Concurrent Version System -> completely obsolete
  – SVN: SubVersioN -> mostly obsolete (should be!)
• Designed by Linus Torvalds (he who gave us Linux!)
• Q: What does ‘git’ stand for?

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Why use git?

• Security
  – Never lose your code again
  – Code is safe against disk failure/earthquakes/meteors

• Convenience
  – Easily deploy your code in several places
  – Easily manage several versions (prod, dev, ...)

• Community
  – Share your code with others
  – Accept bug-fixes & contributions in controlled manner

• Did I mention...
  – Never lose your code again

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Git components

• Command-line interface, the ‘git’ command
• Server ‘hosting’ platforms, web-interface, API
  – Github.com: the original git hosting service
  – Bitbucket.com: used by LBNL/JGI
  – Gitlab.com: recent platform with continuous integration

• Hosting platforms bring added value
  – Issue tracking: bug reports, coupled to git history
  – Wiki: managing documentation
  – Team mgmt: different roles (admin, developer, user)
  – Access mgmt: read/write, read-only, private, public
  – ‘web-hooks’: perform custom actions based on triggers
Git concepts

- **Repository**
  - *Local* or *remote*, a place where git keeps your files
    - On your *local* disk, or on a *remote* server
  
- **Working area**
  - Part of your local repository, you edit your code there

- **Staging area**
  - Part of the local repository where git tracks changes to your working area

- **Branches, tags**
  - Ways to manage sub-groups of files in a repository

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Git workflow

• Change files in your working area

• Tell git about the changes
  – This adds the files to the ‘staging area’
  – At this point, still possible to undo, leaving no trace

• Commit those changes
  – Make them permanent, add them to the repository
  – Now those changes can be recovered, anytime later

• Push the changes to a remote repository
  – Copy your local repository to a remote server
  – Now you have a remote backup

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
More git concepts

• **Clone**
  – A local copy of a remote repository
  – You can change the clone – you own it
  – Access to remote repository controlled by its owner

• **Fork**
  – A remote copy of another remote repository
  – You own the fork, which you can now clone and change

• **A non-concept: ‘The Central Repository’**
  – Git is completely decentralized
  – Can work with multiple remote repositories, simultaneously

• **Confused?**
  – Let’s get stuck into the exercises...

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Git exercises

- **Cookbook approach:**
  - Can cut-&-paste, but better to type in commands yourself

- **Today: do exercises 1, 3, and 4 if you have time**
  - 1) Basic Commit and Tag
  - 2) Undoing Mistakes
  - 3) Using A Remote Repository
  - 4) Using Branches
  - 5) Working in Teams

- **Feel free to work through the rest at your own pace**

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Docker
Docker overview

• **Docker is a ‘container technology’**
  – Linux-specific
    • can’t run Mac OS X, Windows in docker containers
    • But *can* run docker containers on Mac OS X & Windows

• **Similar to virtual machines, but more lightweight**
  – Smaller, faster to start, easier to maintain and manage
  – Lighter on system resources => vastly more scalable

• **Not a virtual machine**
  – Shares the underlying host operating system
  – Less fully isolated from the host => security concerns
  – More of an application-wrapper on steroids

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Docker components

• The ‘docker’ command-line tool
  – A bit of a kitchen-sink, your one-stop shop for everything docker

• The docker-daemon
  – Works behind the scenes to carry out actions
  – Manages container images, processes
  – Builds containers when requested
  – Runs as root, not a user-space daemon

• Docker.com
  – All things docker: installation, documentation, tutorials

• Dockerhub.com
  – Repository of docker containers. Many other repositories exist

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Docker concepts

- **Image**
  - A shrink-wrapped chunk of s/w + its execution environment
- **Image tags**
  - Identify different versions of an image
  - A namespace for separating your images from other peoples
- **Image registry**
  - A place for sharing images with a wider community
  - Dockerhub.com, plus some domain-specific registries
- **Container**
  - A process instantiated from an image
- **Dockerfile**
  - A recipe for building an image: download, compile, configure...
  - Can share either the Dockerfile, or the image, or both

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Docker images: layers

• **Images use the ‘overlay filesystem’ concept**
  – Image is built by adding layers to a base
  – Each command in the Dockerfile adds a new layer
  – Each layer is cached independently
  – Layers can be shared between multiple images
  – Change in one layer invalidates all following layers
    • Forces rebuild (similar to ‘make’ dependencies…)

• **Performance considerations**
  – Too many layers can impede performance
  – Too few can cause excessive rebuilding
  – Building production-quality images takes care, practice

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip
Docker exercises

• Again, a cookbook approach
• Today: 1, 3 and 4 are most interesting
  – 1) Running Images
  – 2) Cleaning up
  – 3) Running a Biobox Container
  – 4) Creating a Docker Image
  – 5) Running on Cori with Shifter

https://bitbucket.org/TWildish/git-docker-tutorial/get/master.zip