

# Using Git and CMake to bring order out of chaos

Oleksandr Moskalenko, Ph.D.  
UFIT Research Computing

Fall 2012 Training Session – November 5<sup>th</sup>, 2012

# Introduction

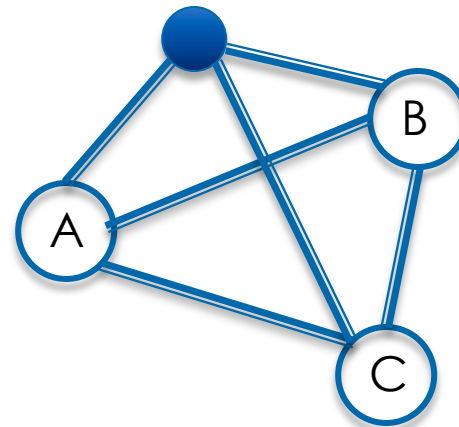
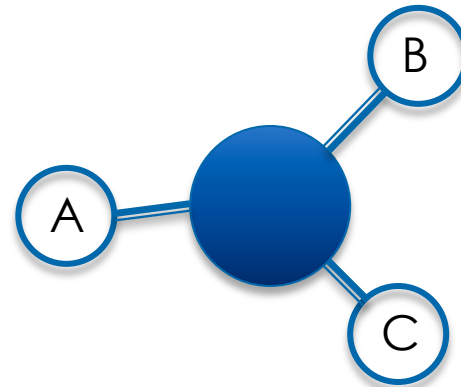
- ▶ Principles of revision control
  - Reasons and methods of revision control
  - Existing systems and Git's place among them
  - Git basics
  - Local and remote repositories, free services
  - The way of collaboration
- ▶ Principles of a build system
  - From the source to the results
  - Overview of the build systems
  - Introduction to Cmake
  - Sample workflow walk-through

# Revision Control Systems

- ▶ Goals:
  - Preserve, display and replay history
    - Undo
    - Backup
    - Track
    - Find problems
  - Enable concurrent development:
    - Version all changes
    - Accept changes from multiple developers
    - Resolve conflicting changes
    - Enable multiple branches of development

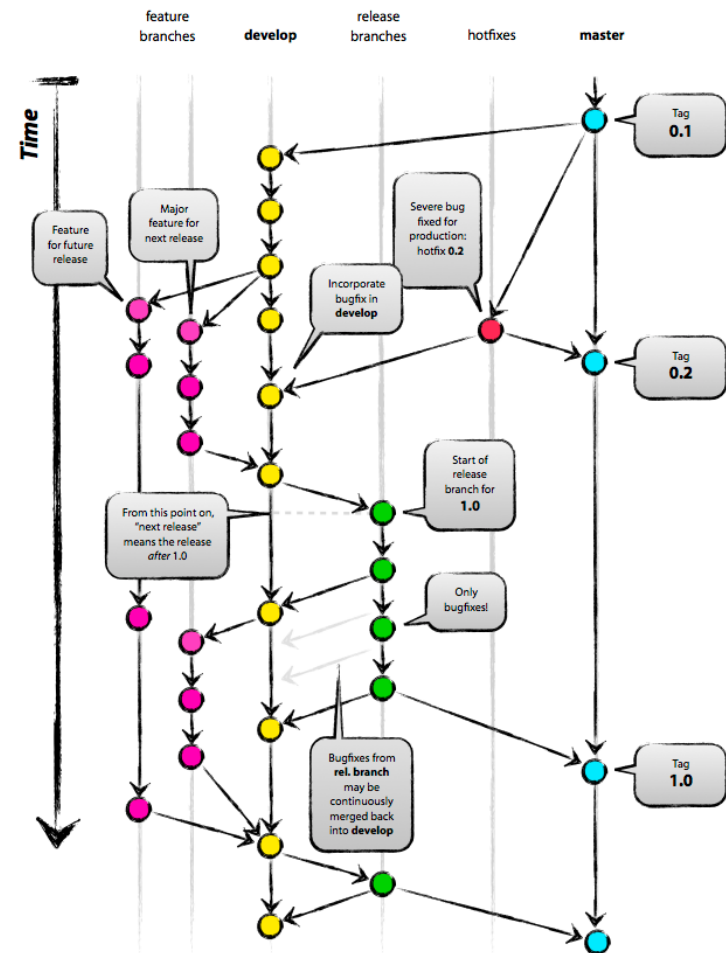
# Types of RCSs

- ▶ Centralized:
  - CVS
  - Subversion
  - Perforce
- ▶ Distributed:
  - Svk
  - Mercurial
  - Bazaar
  - Git



# Git – king of the hill

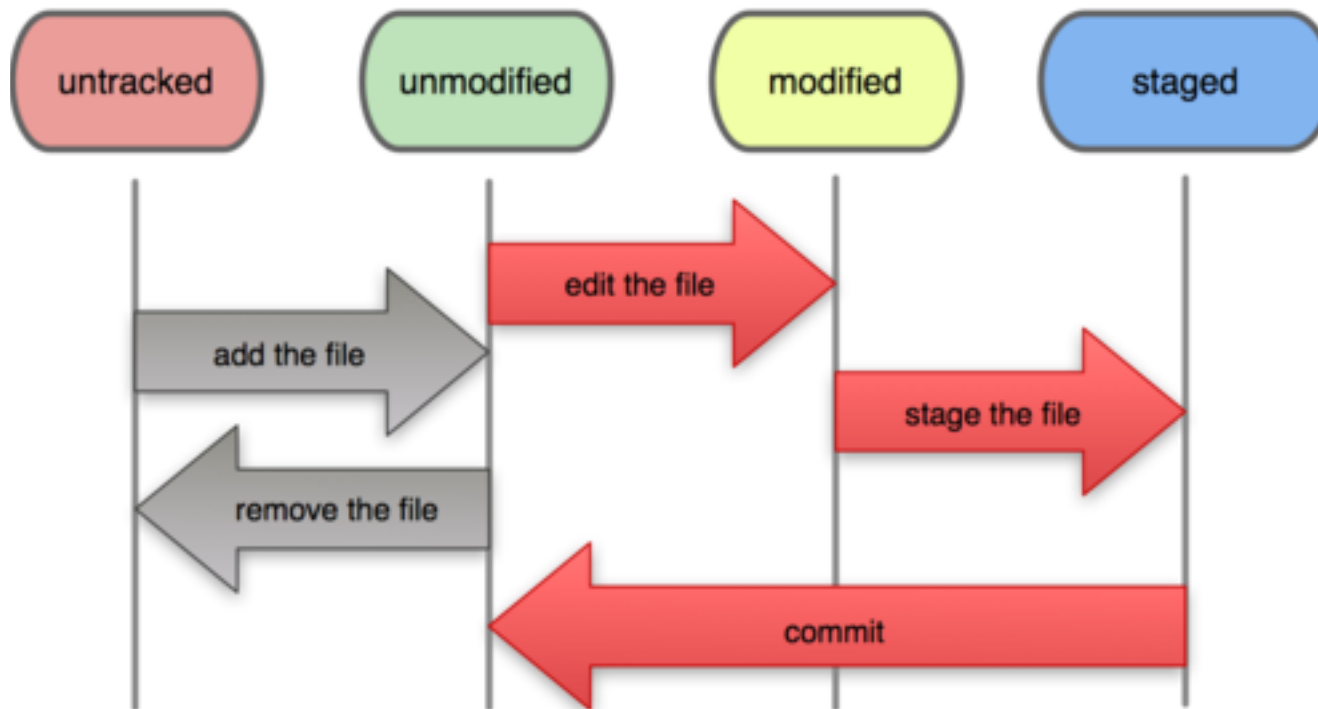
- Distributed:
  - Decentralized but centralized
- Flexible
- Documented
- Scalable
- Supported
- Services
- Branch/merge model



# Revision Control Systems

- ▶ Goals:
  - **Preserve, display and replay history**
    - Undo
    - Backup
    - Track
    - Find problems
  - Enable concurrent development:
    - Version all changes
    - Accept changes from multiple developers
    - Resolve conflicting changes
    - Enable multiple branches of development

# The "Git" cycle



# The “Git” cycle

**Hands on**



# Hosting services

- ▶ Github
  - <https://github.com/>
  - Unlimited free public repositories
  - \$7/Mo/5 private repositories and up
- ▶ Gitorious
  - <http://gitorious.org/>
  - Free public repositories
- ▶ Bitbucket
  - <https://bitbucket.org/>
  - Unlimited private repositories for up to 5 users
  - \$10/Mo/10 users and up

# Hosting Git repositories - DIY

- ▶ Filesystem
  - “git clone file:///scratch/hpc/\$USER/git/project.git”
- ▶ Access-control systems (Gitolite, Gitolite)
  - “git clone [ufrc@git.hpc.ufl.edu:training.git](mailto:ufrc@git.hpc.ufl.edu)”
  - or
  - git remote add ufrc [ufrc@git.hpc.ufl.edu:training.git](mailto:ufrc@git.hpc.ufl.edu)
  - git push

# Git – find problems

- ▶ git bisect start
- ▶ git bisect good 852df3c
- ▶ git bisect bad 185e4a2

Use

- ▶ git bisect good  
and
- ▶ git bisect bad

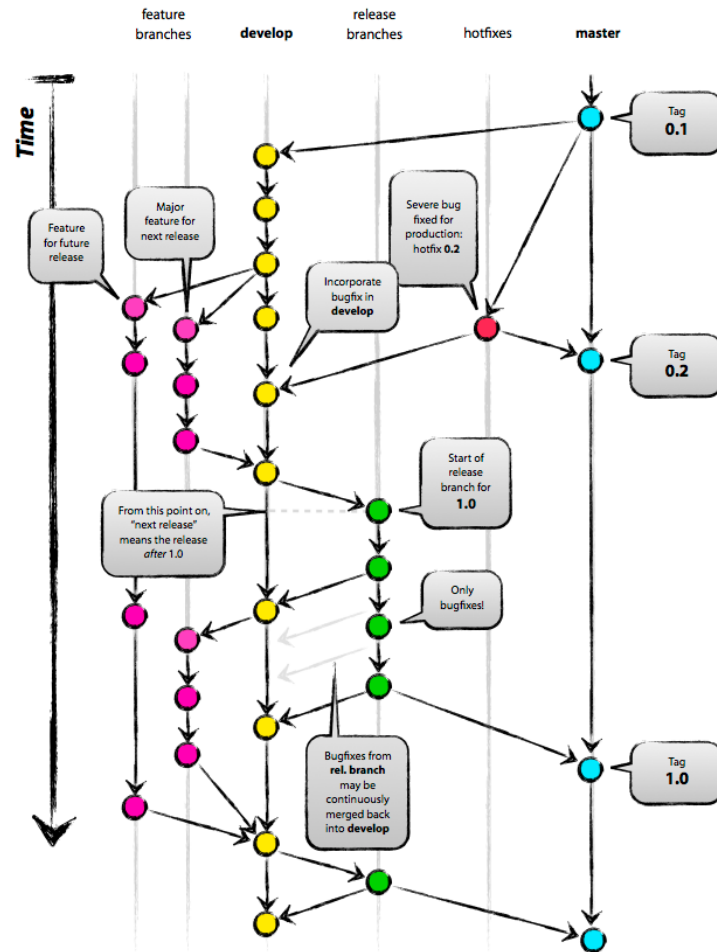
Until the offending revision is located

- ▶ git bisect reset

# Revision Control Systems

- ▶ Goals:
  - Preserve, display and replay history
    - Undo
    - Backup
    - Track
    - Find problems
  - **Enable concurrent development:**
    - Version all changes
    - Accept changes from multiple developers
    - Resolve conflicting changes
    - Enable multiple branches of development

# Collaboration with Git



---

# Collaboration with Git

## Hands on

# Git collaboration - conflicts

- ▶ Strategy depends on activity
  - Infrequent changes:
    - Pull, resolve, commit, push
  - Frequent changes
    - Choose a designated driver who will
    - Pull from other people's repositories
    - Merge and push into the master repository

**Branching is wonderful!**

# GUI Client software - Mac

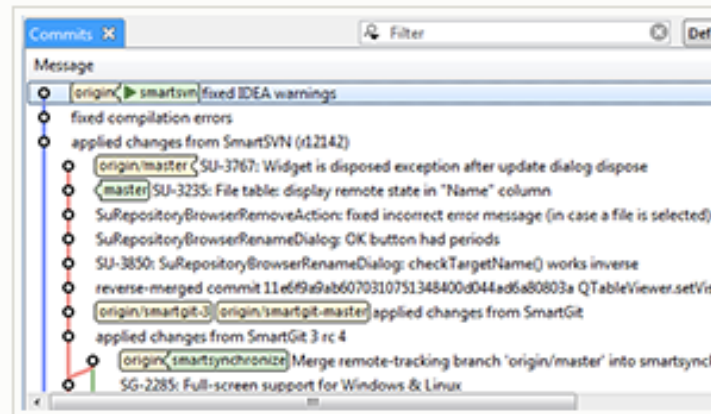
<http://git-scm.com/downloads/guis>

<http://www.sourcetreeapp.com/>

**SourceTree**

**Platforms:** Mac

**Price:** Free

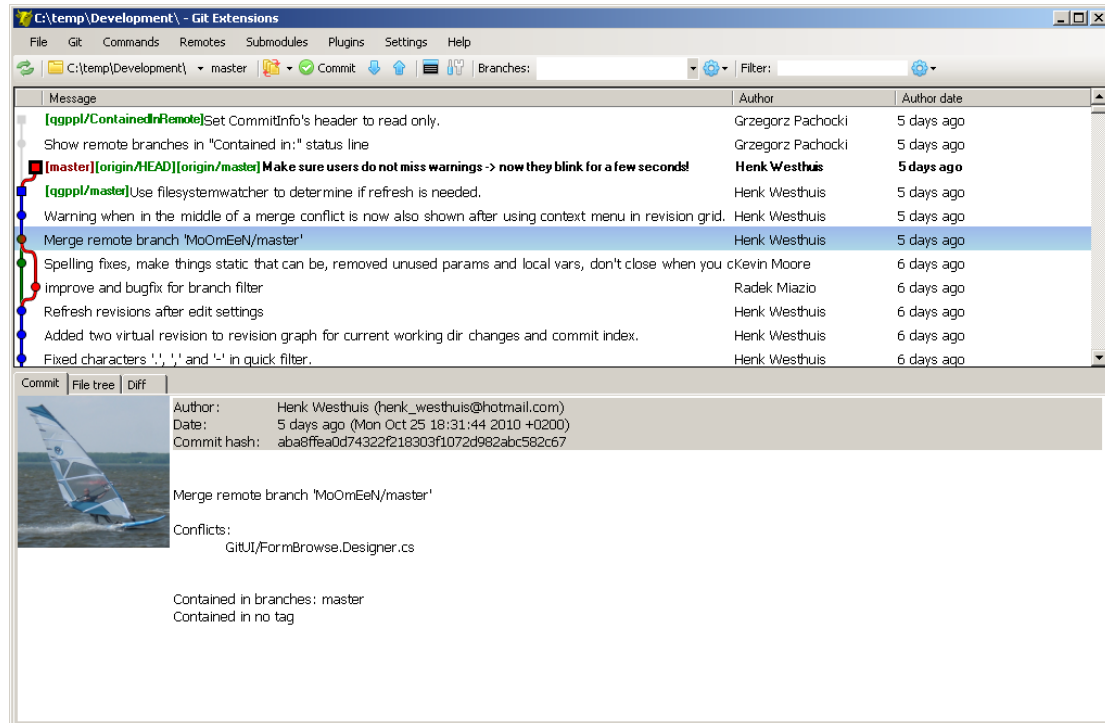




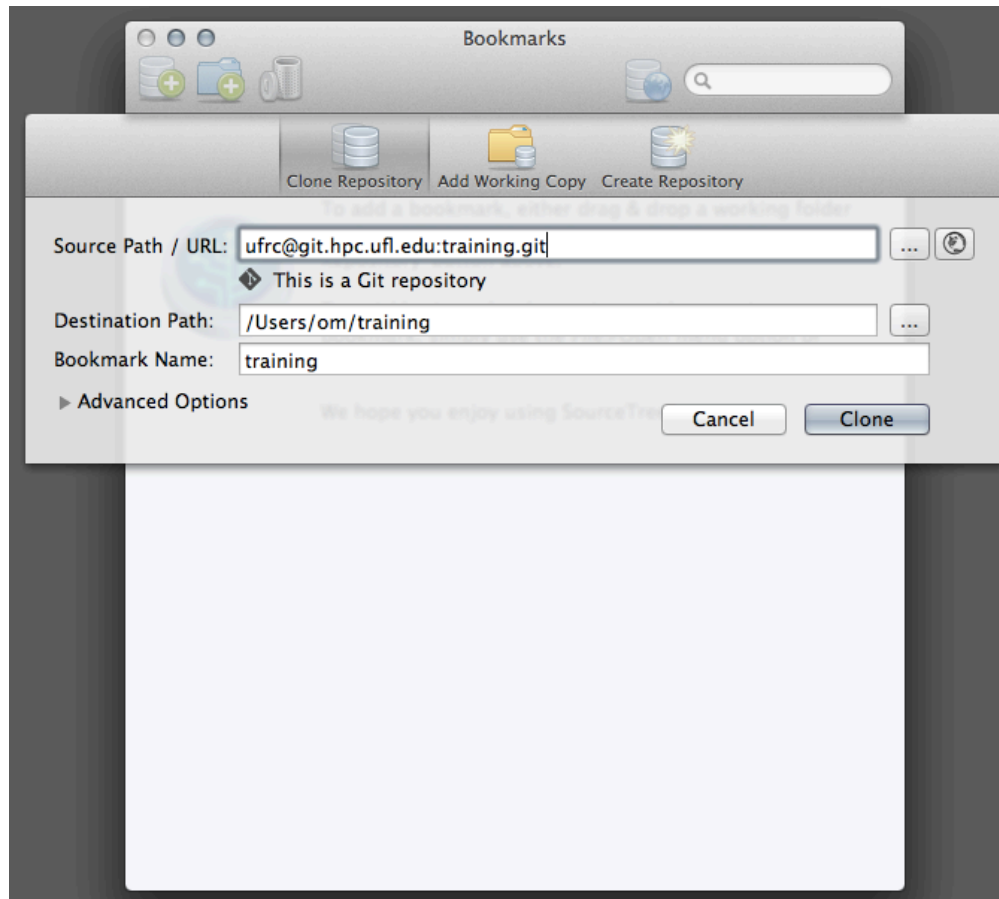
# GUI Client software - Win

<http://git-scm.com/downloads/guis>

<http://git-cola.github.com/> <http://code.google.com/p/gitextensions/>



# GUI Client software - SourceTree



# GUI Client software - SourceTree

The screenshot displays the SourceTree GUI for a Git repository named 'training (Git)'. The interface includes a toolbar with actions like View, Commit, Checkout, Reset, Stash, Add, Remove, Add/Remove, Fetch, Pull, Push, Branch, Merge, Tag, Git Flow, Terminal, and Settings. On the left, a sidebar shows 'FILE STATUS' (Working Copy), 'BRANCHES' (master), 'TAGS', 'REMOTES' (origin), 'STASHES', and 'SUBMODULES'. The main area shows a commit graph and a table of commits. The selected commit is 'Add spaces in comments' by Matt Gitzendanner, with commit hash 185e4a2. Below the commit info, a diff for 'CMakeLists.txt' is shown, highlighting changes in a hunk. The diff shows the addition of a custom command to run code.

Graph	Description	Commit	Author	Date
	origin/master origin/HEAD master Add spaces in comments	185e4a2	Matt Gitzendann...	Nov 5, 2012 5:4...
	Add a comment about the deliverable	cc5e216	Oleksandr Moskal...	Nov 5, 2012 5:3...
	Add a comment about custom commands	423437b	Oleksandr Moskal...	Nov 5, 2012 5:3...
	Add a header comment	852df3c	Oleksandr Moskal...	Nov 5, 2012 5:3...
	Clean up lines	187f22a	Oleksandr Moskal...	Nov 5, 2012 5:2...
	Initial import	63d78e0	Oleksandr Moskal...	Nov 5, 2012 5:2...

**Commit:** 185e4a2954e3d85a31603d6821014fdf8fa33499 [185e4a2]  
**Parents:** cc5e216e84  
**Author:** Matt Gitzendanner <magitz@ufl.edu>  
**Date:** November 5, 2012 5:45:30 AM EST  
**Labels:** HEAD origin/master origin/HEAD master

Add spaces in comments

Filename	Path
CMakeLists.txt	

Context: 3 Lines | Diff Parent | Show Whitespace | External Diff

CMakeLists.txt  
Modified file, 4 lines added, 4 lines removed | Reverse

Hunk 1 : Lines 1-16

```
1 - #Sample file for UF HPC Training on 2012-11-05
1 + # Sample file for UF HPC Training on 2012-11-05
2 2 project (
3 3
4 - #Use a custom command to run your code
4 + # Use a custom command to run your code
5 5 set (input_data_file ${PROJECT_SOURCE_DIR}/data.dat)
6 6 set (output_c_file data.c)
7 7 add_custom_command (
8 8 OUTPUT ${output_c_file}
9 - COMMAND /tools/bin/make-data-file
9 + COMMAND /project/bio/training/2012-11-05/scripts/fasta_to_tabular
10 10 < ${input_data_file}
11 11 > ${output_c_file}
```

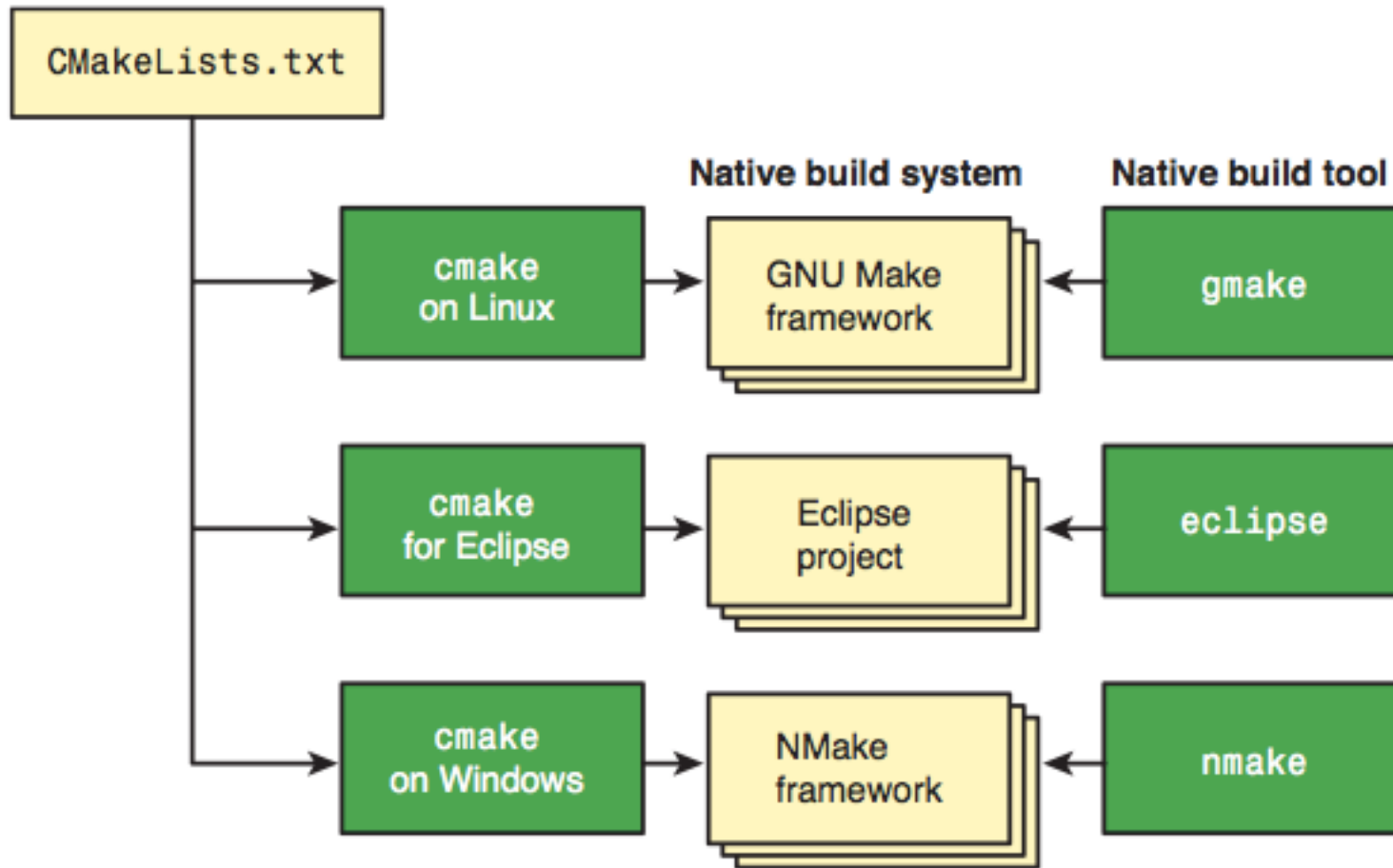
Atlassian

# Build Systems

- ▶ Goal:
  - Convert data from inputs into outputs
  - Software development
    - Compile source code into executable binaries
    - Package scripts
    - Build and package web applications
    - Run tests on the code
    - Run analysis tools
    - Generate documentation

**Sounds familiar?**

# CMake workflow overview



Peter Smith, Software build systems: Principles and Experience, 2011

# CMake Usage - Introduction

- ▶ Contains all directives for building your project
- ▶ Allows in and out of tree builds
- ▶ Creates a native build system i.e.
- ▶ “make” on Linux, “nmake” on Windows etc

# CMake Usage – directory structure

- ▶ /project/bio/training/2012-11-05:
  - CMakeLists.txt
  - bin
  - input
  - output
  - work

Append path to the “bin” directory to \$PATH in the environment or use “FIND\_PROGRAM” commands in CMakeLists.txt

# CMake Usage – run commands

- ▶ mkdir output
- ▶ cd output
- ▶ cmake ..
- ▶ make



# CMakeLists.txt sample

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8.0)
PROJECT (MetagenomicsStart)
SET(CMAKE_PREFIX_PATH ${CMAKE_SOURCE_DIR})

FIND_PROGRAM(TRIM_SEQ short_reads_trim_seq.py)
ADD_CUSTOM_COMMAND(
  COMMENT "Select high-quality sequences"
  OUTPUT ${CMAKE_SOURCE_DIR}/work/high_quality.fa
  COMMAND ${TRIM_SEQ} 20 50 ${CMAKE_SOURCE_DIR}/work/high_quality.fa $
${CMAKE_SOURCE_DIR}/input/input.fa ${CMAKE_SOURCE_DIR}/input/input.qual ye + s
  DEPENDS ${CMAKE_SOURCE_DIR}/input/input.fa
)
ADD_CUSTOM_TARGET(high_qual_fa ALL DEPENDS ${CMAKE_SOURCE_DIR}/work/
high_quality.fa)

/and so on/
```

# CMake Run

**Scanning dependencies of target high\_qual\_fa**

[ 6%] **Select high-quality sequences**

[ 6%] **Built target high\_qual\_fa**

**Scanning dependencies of target high\_qual\_read\_fa**

[ 13%] **Convert HQ fasta to tab**

[ 20%] **Add 'readXX' column**

[ 26%] **Convert tab to fasta**

[ 33%] **Built target high\_qual\_read\_fa**

**Scanning dependencies of target high\_qual\_read\_len**

[ 40%] **Compute length**

[ 66%] **Built target high\_qual\_read\_len**

**Scanning dependencies of target high\_qual\_tab**

[ 80%] **Built target high\_qual\_tab**

**Scanning dependencies of target high\_qual\_tab\_read**

[100%] **Built target high\_qual\_tab\_read**

# CMake Outputs

**high\_quality.fa**  
**high\_quality\_read.fa**  
**high\_quality\_read.len**  
**high\_quality\_read.tsv**  
**high\_quality.tsv**

# CMake – to bash or not to bash?

- A simple build looks like more complex than an equivalent shell script
- Advantages:
  - More consistent rules when pipelines get bigger
  - Automatically provided variables
  - Build proceeds along a dependency graph
  - If a step fails – subsequent builds will continue from the correct step automatically
  - Out of tree builds are clean automatically
  - Portable – native build system generation
  - ?

---

# Questions?

---

Thank you?