Introduction to Software Project Management

On UD HPC Community Clusters

William Totten
Network & Systems Services

Aspects of software projects

- Code management
  - Using revision control to track changes
  - Collaboration
  - Accessing different versions of code
  - Peer review
- Build management
  - Creating a Makefile to simplify building
  - Optional Features
  - Supporting more than one OS or system configuration
  - GNU Autoconf
  - CMake

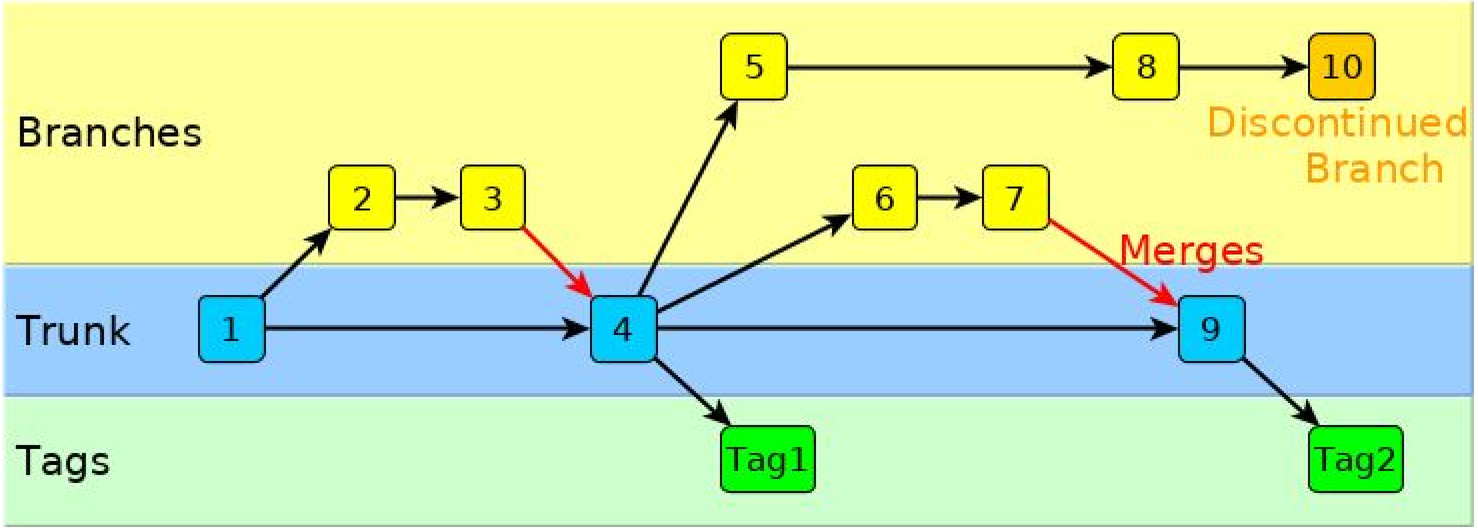
Revision control systems

- Git
    - Local repositories
    - Consider https://github.com/ (free for open source) or https://gitlab.com/ (free for all)
    - Allows for multiple local commits before uploading them centrally
- Subversion (svn)
    - Local repositories
    - Consider https://sourceforge.net/
    - You can check out any part of the tree you want
- Others
    - RCS
    - CVS
    - Mercurial (aka hg)
    - ...

Revision control terminology

branch	A set of files under version control may be *branched* or *forked* at a point in time so that, from that time forward, two copies of those files may develop at different speeds or in different ways independently of each other.
checkout	To *clone*, *check out* (or *co*) is to create a local working copy from the repository. A user may specify a specific revision or obtain the latest. The term 'checkout' can also be used as a noun to describe the working copy.
commit	To *commit* (*check in*, *ci* or, more rarely, *install*, *submit* or *record*) is to write or merge the changes made in the working copy back to the repository. The terms 'commit' and 'checkin' can also be used as nouns to describe the new revision that is created as a result of committing.
conflict	A conflict occurs when different parties make changes to the same document, and the system is unable to reconcile the changes. A user must *resolve* the conflict by combining the changes, or by selecting one change in favour of the other.
merge	A *merge* or *integration* is an operation in which two sets of changes are applied to a file or set of files.
pull/push	Copy revisions from one repository into another. Pull is initiated by the receiving repository, while push is initiated by the source. Fetch is sometimes used as a synonym for pull, or to mean a pull followed by an update.
revision	Also version: A version is any change in form.
tag	A *tag* or *label* refers to an important snapshot in time, consistent across many files. These files at that point may all be tagged with a user-friendly, meaningful name or revision number.
trunk	The unique line of development that is not a branch (sometimes also called Baseline, Mainline or Master)

Revision control in action



- In SVN, revisions are numbers like shown above and tags can be any text string
- If this were Git, the revisions would have seemingly random hexadecimal strings like f7fd3d4

Local SVN

- In SVN, the repository should go someplace special
- You may want to keep an "official" checkout in a workgroup directory
- You can start out with an empty repository, then add stuff
- Try to make the names match up between the repository and checkout
- You have to manually add the files you want in the repository
- Once you check-in the files, anyone else with access to the repository can create their own working copy

```
$ mkdir repos
$ svnadmin create repos/project1
$ svn mkdir -m Structure file:///home/1474/spm/svn_example/r
epos/project1/{trunk,branches,tags}
Committed revision 1.
$ svn co file:///home/1474/spm/svn_example/repos/project1/tr
unk project1
Checked out revision 1.
$ cd project1
$ cp /opt/templates/dev-projects/C_Executable/* .
$ svn add *
A         Makefile
A         helloworld.c
A         printmsg.c
A         printmsg.h
$ svn ci -m 'Initial code check-in'
Adding         Makefile
...
Transmitting file data ....
Committed revision 2.
```

Updating code in an SVN repository

- Always try to ensure your copy is up-to-date before making changes
- Use whatever process you are most comfortable with to change files
- Always use `svn add`, `svn rm`, and `svn mv` to add/remove/move files
- You should always provide a comment when making changes
- SVN is centralized, so version numbers are monotonically increasing
- SVN can tell you if any new files aren't revision controlled

```
$ svn up
At revision 2.
$ vi Makefile
$ svn ci -m 'Simplify Makefile, switch to gcc'
Sending        Makefile
Transmitting file data .
Committed revision 3.
$ make
gcc -g -O   -c -o helloworld.o helloworld.c
gcc -g -O   -c -o printmsg.o printmsg.c
gcc -g -O -o helloworld helloworld.o printmsg.o  -lm
$ svn status
?       helloworld
$
```

Local Git

- Git is distributed, so every clone is a repository by definition
- You may want to keep an "official" clone in a workgroup directory
- You can start out with an empty repository, then add stuff
- Git is distributed, so it really wants to know something unique about you
- You have to manually add the files you want in the repository
- Once you check-in the files, anyone else with access to the repository can clone their own copy

```
$ mkdir project1
$ cd project1
$ git init .
Initialized empty Git repository in
/home/1474/spm/git_example/project1/.git/
$ git config --global user.email totten@udel.edu
$ cp /opt/templates/dev-projects/C_Executable/* .
$ git add *
$ git commit -am 'Initial code check-in'
[master (root-commit) f7fd3d4] Initial code check-in
 4 files changed, 153 insertions(+), 0 deletions(-)
 create mode 100644 Makefile
 create mode 100644 helloworld.c
 create mode 100644 printmsg.c
 create mode 100644 printmsg.h
```

Using github.com

- Get a login for yourself on github.com, I recommend adding some SSH keys
- Create the new repository at https://github.com/new
- Now clone a working repository for yourself
- Git is distributed, so it really wants to know something unique about you
- You have to manually add the files you want in the repository
- Once you check-in the files, anyone else with access to the repository can clone or pull their own copy
- The project must be Open Source

```
$ git clone git@github.com:biell/project1.git
Initialized empty Git repository in
/home/1474/spm/git_example/project1/.git/
remote: Counting objects: 4, done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 4
Receiving objects: 100% (4/4), 850 B, done.
$ cd project1
$ git config --global user.email totten@udel.edu
$ vi README.md
$ cp /opt/templates/dev-projects/C_Executable/* .
$ git add *
$ git commit -am 'Initial code check-in'
[master (root-commit) 38ad2e3] Initial code check-in
 5 files changed, 153 insertions(+), 0 deletions(-)
 create mode 100644 Makefile
 create mode 100644 helloworld.c
 create mode 100644 printmsg.c
 create mode 100644 printmsg.h
```

Updating code in a Git repository

- Pull the latest trunk when using github
- Use whatever process you are most comfortable with to change files
- Always use `git add`, `git rm`, and `git mv` to add/remove/move files
- You can choose what to commit, add the "-a" to commit all changes.
- You commit to your local, distributed copy, not a central repository
- You can push any number of commits all at once

```
$ git pull
Already up-to-date.
$ vi Makefile
$ git commit -am 'Simplify Makefile, switch to gcc'
[master 5c0bfbd] update Simplify Makefile, switch to gcc
1 file changed, 3 insertions(+), 3 deletions(-)
$ git push
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 416 bytes | 0 bytes/s, done.
Total 4 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local
objects.
To ssh://github.com/biell/project1.git
   38ad2e3..5c0bfbd  master -> master
$ make
...
$
```

github.com is great for scientific research

- https://github.com/gromacs/gromacs
- https://github.com/numpy/numpy
- https://github.com/FFTW/fftw3
- https://github.com/JuliaLang/julia
- https://github.com/opencollab/scilab

Using make to automate builds

- make is the primary tool of software developers to manage software builds
- make looks for a file named "Makefile" and follows its instructions
  - other file names (e.g. "makefile" and "GNUmakefile") also work
- make ensures the software is compiled the same way each time
- make can be told about dependencies, and ensure they compile first
- make figures out which files have changed and only re-compiles what is necessary
- make is "setup and forget", you don't have to become an expert.
- You can start with a similar project's Makefile, and modify it to suit your needs

Using make to automate builds

```
# Makefile for helloworld

# programs to use
CC = gcc
RM = rm

# define any compile-time flags
CFLAGS = -g -O

# Libraries
LIBS = -lm

# define the C source files
SRCS = printmsg.c helloworld.c

# define the C object files (We could have just written "printmsg.o helloworld.o" here)
OBJS = $(SRCS:.c=.o)

all: $(OBJS)
	$(CC) $(CFLAGS) -o helloworld $(OBJS) $(LIBS)

.c.o:
	$(CC) $(CFLAGS) $(INCLUDES) -c $<  -o $@

clean:
	$(RM) *.o helloworld
```

Dynamic Builds

- make is all you need when all builds always use the same:
  - Compiler
  - Libraries
  - OS
  - Integer and/or floating-point precision
- If your changes are really simple for different builds, you can put variables near the front of your Makefile
- Don't write your own build process
- When you want to introduce more complex options and different build types, it is time to consider getting some help for make
  - GNU Autoconf
  - CMake

What is GNU Autoconf

- Autoconf is used by programmers to create a script called “configure”
- It is designed to support variants of C, Fortran, and Erlang
- Configure looks at the system and figures out how it works
- Configure writes a file “config.h” out for programmers to use
- Configure writes a “Makefile” for you to build the software
- You compile and install the software using the program “make”

```
$ PREFIX=/home/work/lab/sw/pkg/1.3
$ ./configure --help | more
 ...
$ ./configure --prefix=$PREFIX
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
 ...
config.status: creating Makefile
config.status: creating config.h
config.status: executing depfiles commands
config.status: executing libtool commands
$ make
 ...
$ make test
 ...
$ make install
 ...
$
```

What is CMake

- CMake is used by developers to manage the software build process natively on UNIX like and Windows operating systems
- It is designed to be extensible beyond C and Fortran
- CMake writes a “Makefile” for you to build the software (on Unix)
- You compile and install the software using the program “make” (on Unix)
- A variety of build environments are supported on Windows
- CMake has a menu system to help select build parameters "`ccmake`"

```
$ PREFIX=/home/work/lab/sw/pkg/1.3
$ mkdir build; cd build
$ cmake -DCMAKE_INSTALL_PREFIX=$PREFIX ..
-- The C compiler identification is GNU 4.4.7
-- The CXX compiler identification is GNU 4.4.7
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
 ...
-- Configuring done
-- Generating done
-- Build files have been written to: /home/work/lab/...
$ make
 ...
$ make test
 ...
$ make install
 ...
$
```

Getting Started

SVN	• http://maverick.inria.fr/Members/Xavier.Decoret/resources/svn/index.html
Git	• https://try.github.io/ • https://git-scm.com/docs/gittutorial
make	• http://mrbook.org/blog/tutorials/make/ • http://www.tutorialspoint.com/makefile/
GNU Autoconf	• http://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html
CMake	• https://cmake.org/cmake-tutorial

Questions and Open Forum

University of Delaware

Information Technology
Research Computing

Let's try git out

https://try.github.io/