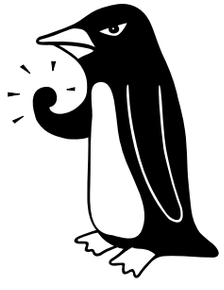
Version Control for Fun and Profit

Chris Brady
Heather Ratcliffe

“The Angry Penguin“, used under creative commons licence
from Swantje Hess and Jannis Pohlmann.



Warwick RSE

12/12/2017

Outline

- Part 1 - Motivations and background
  - Overview
  - Why use version control?
  - How did we get here?
- Part 2 - Using git for version control

Part 1 - Motivations and Background

Overview

- Version control
  - Record changes that you make to a file or system of files
  - Allows you to keep a log of why/by whom those changes were made
  - Allows you to go back through those changes to get back to old versions
  - Help deal with merging incompatible changes from different sources
- Similar term “Source Code Management”

Why use version control?

- "I didn't mean to do that!"
  - Can go back to before you made edits that haven't worked
- "What did this code look like when I wrote that?"
  - Can go back as far as you want to look at old versions that you used for papers or talks
- "How can I work on these different things without them interfering?"
  - Branches allow you to work on different bits and then merge them at the end

Why use version control?

- "I want a secure copy of my code"
  - Most version control systems have a client-server functionality. Can easily store an offsite backup.
  - Many suitable free services, and can easily set up your own
- "How do I work with other people collaboratively?"
  - Most modern version control systems include specific tools for working with other people.
  - There are more powerful tools to make it even easier too

Why use version control?

- Lots of tools out there
- Most likely you're going to be using git (https://git-scm.com)
- Quite likely going to be using the most popular public service, GitHub (https://github.com)
- SCRTP at Warwick has it's own git system (https://wiki.csc.warwick.ac.uk/twiki/bin/view/Main/GitServer)

Why NOT use version control?

- You definitely can! If …
  - Working on your own
  - Mainly developing one feature at a time
  - Keep careful offsite backups
  - Keep separate copies of every version of the code that you use for anything
- Require more effort and discipline than using version control nowadays

How did we get here?

- Version control is literally as old as computers
- United States National Archives Records Service punch card storage warehouse in 1959
- ~100MB / Forklift pallet
- Stored both programs and data
- Important programs would be kept in archives and repunched when changed
- Old versions kept for some time

How did we get here?

- Source Code Control System (SCCS) - 1972 - 2015
  - Contains
    - Original version of each file
    - List of changes to each file
    - Permission flags
- Uses filesystem for all locking
- Still technically part of the UNIX specification

How did we get here?

- Revision Control System (RCS) - (1982 - 2015)
  - Quite important improvements over SCCS at the time, but looks very incremental to modern eyes
- Concurrent Versions System (CVS) - (1990 - 2008)
  - Client - Server RCS
  - Has merging mechanism to allow multiple people to work on file at once
  - First true *branches* - can develop variants of a code in same repository

How did we get here?

- Subversion - (2000 - 2017 last release, still active)
  - Same client - server model as CVS, but no remaining links to RCS
  - Moved/renamed/copied/deleted files now retain version history
  - Commits are atomic (either succeed completely or fail to do anything)
  - Automated merging between branches
    - Makes developing of features a lot easier

How did we get here?

- Bitkeeper (2000 - 2017 (last release, still active))
  - Originally closed source, commercial software (opened in 2016)
  - Same basic features as Subversion
  - Used a distributed storage system
    - Each developer has as much of the repository as wanted in a local copy
    - Only merges onto central server when ready
- Used for Linux Kernel from 1999 until 2006 using free “community edition”. Then this vanished
- https://lwn.net/Articles/130746/

How did we get here?

- Linus Torvalds responded by writing
- Git (2005 - 2017 (last release, still active))
  - Same distributed model
    - Push to remote server when needed (if wanted)
    - Very sophisticated branching and merging system
    - Become very popular because of GitHub
    - GIT IS NOT GITHUB!

How did we get here?

- There are many, many others
- Most likely to meet
  - Mercurial (https://www.mercurial-scm.org)
    - Also written in response to BitKeeper, but not used by Linux kernel
  - Bitkeeper
    - Now open source!
  - Subversion
    - Still around, and still good
  - Visual Studio Team Services
    - Microsoft tools only, but quite good

Part 2 - Using git for version control

Create a repository

```
Marlow:demo bradyc$ git init
Initialized empty Git repository in /Users/bradyc/demo/.git/
Marlow:demo bradyc$
```

- Simply type “git init”

- Directory is now a working git repository

- Be careful about creating a git repository in a directory that isn’t the bottom of the directory tree!

Designate files for repository

```
Marlow:demo bradyc$ mkdir src
Marlow:demo bradyc$ touch src/wave.f90
Marlow:demo bradyc$ git add src/
Marlow:demo bradyc$
```

- Create a directory and put a file in it
- "git add src/" tells git to put the directory src and all files within it under version control
    - Not yet actually in the repository!
- I'm using Fortran because I'm a physicist
    - Works pretty well with almost any text based file
        - Best with things like C/C++/Fortran/Python that it understands
        - Can now work with Jupyter notebooks without showing you all of the guts

Add files to the repository

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   src/wave.f90
#
```

- "git commit" will actually add the file to the repository

  - Will open an editor to specify a "commit message"

    - I'm using Vim. Default will depend on your system

  - Generally git commit messages should follow standard format

Git commit message

```
First check in of wave.f90

wave.f90 will be a demo of using a "wave" type MPI cyclic transfer
0->1->2->3->4->0 etc. in order. This is inefficient and it shown
merely for teaching purposes
```

- First line is the subject. Keep it to <= 50 characters
- Second line should be blank
- Subsequent lines are the “body” of the message
- Should limit body lines to <=72 characters
- As many as you want, but be concise

After writing message

```
[master (root-commit) 750edb5] First check in of wave.f90
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 src/wave.f90
Marlow:demo bradyc$
```

- When you save the file and exit your editor git will give you a summary of what's just happened
  - In this case, it's created the file "wave.f90" as I wanted it to
- If you quit your editor without saving this cancels the commit
- "wave.f90" is now under version control, and I can always get back to this version

Editing wave.f90

```
PROGRAM wave

  USE mpi
  IMPLICIT NONE

  INTEGER, PARAMETER :: tag = 100

  INTEGER :: rank, recv_rank
  INTEGER :: nproc
  INTEGER :: left, right
  INTEGER :: ierr

  CALL MPI_Init(ierr)

  CALL MPI_Comm_size(MPI_COMM_WORLD, nproc, ierr)
  CALL MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr)

  !Set up periodic domain
  left = rank - 1
  IF (left < 0) left = nproc - 1
  right = rank + 1
  IF (right > nproc - 1) right = 0

  IF (rank == 0)
    CALL MPI_Send(rank, 1, MPI_INTEGER, right, tag, MPI_COMM_WORLD, ierr)
    CALL MPI_Recv(recv_rank, 1, MPI_INTEGER, left, tag, MPI_COMM_WORLD, &
        MPI_STATUS_IGNORE, ierr)
  ELSE
    CALL MPI_Recv(recv_rank, 1, MPI_INTEGER, left, tag, MPI_COMM_WORLD, &
        MPI_STATUS_IGNORE, ierr)
    CALL MPI_Send(rank, 1, MPI_INTEGER, right, tag, MPI_COMM_WORLD, ierr)
  END IF


  CALL MPI_Finalize(ierr)

END PROGRAM wave
```

Adding the changes

```
Marlow:demo bradyc$ git commit
On branch master
Changes not staged for commit:
        modified:   src/wave.f90

no changes added to commit
Marlow:demo bradyc$
```

- Not just “git commit” again!
- That tells me that I have a modified file, but it isn’t “staged for commit”
- Have to “git add” it again, then “git commit”
- Can have as many adds as you want before a commit. That is “staging” the files
- Slightly risky alternative “git commit -a” commits everything changed since last commit

Adding the changes

```
[master 867a375] Added content to wave.f90
 1 file changed, 37 insertions(+)
```

- Once again editor comes up
- Same commit message format
- Should describe the changes that you have made
- On saving the file in the editor see the same commit summary
  - Now telling me that it’s added 37 lines

Showing the log

```
Marlow:demo bradyc$ git log
commit 867a3759bfd3afddcb6e3ba1c562c312ec1a87bd
Author: Chris Brady [redacted]
Date:   Mon Nov 13 14:50:35 2017 +0000

    Added content to wave.f90

    Wave.f90 is now an example of how not to do MPI in Fortran

commit 750edb57a465acfb5dd19050706cd738e4a12a7d
Author: Chris Brady [redacted]
Date:   Mon Nov 13 14:29:46 2017 +0000

    First check in of wave.f90

    wave.f90 will be a demo of using a "wave" type MPI cyclic transfer
    0->1->2->3->4->0 etc. in order. This is inefficient and it shown
    merely for teaching purposes
```

- Can see the list of commit messages using “git log”
- Note the string after the word “commit”. It is the commit ID.
  - This uniquely identifies a given commit

Seeing differences

- Using the command “git diff” followed by a commit ID shows you the changes between the current state of the code and the one referred to in the by the commit ID

- Adding a list of filenames at the end allows you to see the differences in only specific files

- The result of the command is in “git-diff” format
  - Lines with a + have been added since the specified commit
  - Lines with a - have been removed
  - Lines without a symbol are only there for context and are unchanged

git diff output

```
diff --git a/src/wave.f90 b/src/wave.f90
index ffaa053..c2e694e 100644
--- a/src/wave.f90
+++ b/src/wave.f90
@@ -3,7 +3,7 @@ PROGRAM wave
    USE mpi
    IMPLICIT NONE

-   INTEGER, PARAMETER :: tag = 100
+   INTEGER :: dummy_int

    INTEGER :: rank, recv_rank
    INTEGER :: nproc
```

- Example git diff
  - I have removed the key line referring to “tag”
  - and replaced it with “dummy_int”

Reverting to undo bad changes

- Undoing changes in git can be a mess
  - Distributed system, so if code has ever been out of your control you can't just go back
    - Reverts are in general simply changes that put things back to how they used to be
    - Git log will show original commits and reverts
- Command is "git revert"

git revert

```
Marlow:demo bradyc$ git revert 867a3759bfd3afddcb6e3ba1c562c312ec1a87bd..770049b
311aa55bec3b254cdd3a340219dbdb43c
[detached HEAD 1c4a2b8] Revert "Similar addition"
 1 file changed, 1 deletion(-)
[detached HEAD 406fb63] Revert "Deliberated removed key line for teaching purpos
e"
 1 file changed, 1 insertion(+)
```

- Lots of flexibility, but mostly you want to do
  - git revert {lower_bound_commit_id}..{upper_bound_commit_id}
- Lower bound is exclusive
- Upper bound is inclusive

git revert

- When git revert operates, it creates a new commit undoing each commit that you want to revert

- You get an editor pop-up for each with a default message that says

- Revert “{original commit message}”

- No real need to change them

git branch

- If you are working on multiple features then branches are useful
- Branches are code versions that git keeps separate for you
- Changes to one branch do not affect any other
- There is a default branch called “master” created when you create the repository
- A git repository is always working on one branch or another (sometime a temporary branch, but ignore this here)
- Adds and commits are always to the branch that you are working on

git branch

```
Marlow:demo bradyc$ git branch version2
Marlow:demo bradyc$ git branch
* master
  version2
```

- To create a branch, just type “git branch {name}”

- A new branch is created based on the last commit in the branch that you are on

- Simply creating a branch does not move you to it. You are still exactly where you are before

- You can check what branch you are on by typing “git branch” with no parameters

git checkout

```
Marlow:demo bradyc$ git checkout version2
Switched to branch 'version2'
```

- To move between branches, you use "git checkout {branch_name}"

- This will tell you that it has switched to the named branch if it has managed to do so

git checkout

```
Marlow:demo bradyc$ git checkout -b newbranch version2
Switched to a new branch 'newbranch'
Marlow:demo bradyc$ git branch
  master
* newbranch
  version2
```

- You can create a new branch based off an existing branch and check it out in a single command using

- "git checkout -b {new_branch_name} {existing_branch_name}"

- This is very useful when you're working with remote branches (i.e. those you get from a git server)

git checkout

```
Marlow:demo bradyc$ git checkout -bold_version 750edb57a465acfb5dd19050706cd738e
4a12a7d
Switched to a new branch 'old_version'
```

- Sometimes you want to go back to an old version of the code

- "git checkout -b{new_branch_name}{commit ID}"

- This checks out the code in the state that it was in at the specified commit ID.

- If you don't specify a branch, the repository goes into "detached head mode" - temporary branch

  - Probably best to not do it

Checking branches

- You can check that branches are working as you expect by
  - Creating a branch
  - Changing to the branch
  - Making some changes
  - Adding them using “git add” and “git commit”
  - Changing back to master
- Your changes to your branch will not have appeared in master

Changing branches

```
Marlow:demo bradyc$ git checkout master
error: Your local changes to the following files would be overwritten by checkou
t:
        src/wave.f90
Please, commit your changes or stash them before you can switch branches.
Aborting
```

- Once branches have changed relative to each other you can no longer carry changes between them

- If you make changes in a branch and then try to move to another branch, without committing the changes you will get an error message

- Either

  - commit the changes in the branch that you are on

  - use git-stash (https://git-scm.com/docs/git-stash)

Bringing branches back

```
Marlow:demo bradyc$ git merge version2
Updating 867a375..b0f854a
Fast-forward
 src/wave.f90 | 1 +
 1 file changed, 1 insertion(+)
```

- If you're using branches to develop features (a very common way of working) you'll want to bring them back together to form a single version with all the features

- Termed "merging"

- "git merge {other_branch_name}" brings the other branch's content into this branch

- If you're lucky, you'll see what's at the top and the merge is automatic

Manual Merge

- If git can't work out how to combine the changes between the version then it'll just put diff formatted markers into the file to say what's changed and where

- You have to go through and remove these markers, leaving a single working version of the code

- Commit the finished version using "git commit" as normal (or "git merge -- continue" in newer versions of git)

- There are tools to help, but it's never fun

Git remote server

- Git is a distributed, networked version control system.
- Has commands to control this
- Collectively called “git remote” commands
- You can clone a remote repository and it remembers that it’s attached to that remote
- A local repository can be told that it’s a local copy of an remote repository

git clone

```
Marlow:demo2 bradyc$ git clone https://github.com/LMFDB/lmfdb.git
Cloning into 'lmfdb'...
remote: Counting objects: 46154, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 46154 (delta 0), reused 2 (delta 0), pack-reused 46148
Receiving objects: 100% (46154/46154), 19.45 MiB | 1.52 MiB/s, done.
Resolving deltas: 100% (34294/34294), done.
Checking connectivity... done.
Marlow:demo2 bradyc$
```

- To clone a remote repository, you need to have a URL for the remote server
  - This is a github repository, so big green button
- Command is then "git clone {remote_url}"
- No need to do "git init" or anything else
- Creates new functioning local repository in a subdirectory of where you ran the command

git branch -a

```
Marlow:lmfdb bradyc$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/beta
  remotes/origin/edgarcosta-patch-1
  remotes/origin/id-artin-det
  remotes/origin/master
  remotes/origin/nf-completeness
  remotes/origin/prod
Marlow:lmfdb bradyc$
```

- Running “git branch -a” also tells you about remote branches
- Once again, there exists a “master” branch, which is now a local reference to “remotes/origin/master”
- You do not by default have copies of all of those remote branches
- You get them using “git checkout -b”

git pull

```
Marlow:lmfdb bradyc$ git pull
Updating 035b316..7cb3e37
Fast-forward
 lmfdb/lfunctions/LfunctionDatabase.py | 9 +--------
 1 file changed, 1 insertion(+), 8 deletions(-)
Marlow:lmfdb bradyc$
```

- If you have a copy of a repository that is less recent than the version on the remote server you can update it using "git pull"
- This can happen
  - Because you've changed the code on another machine
  - Another developer has updated the server version
  - Git doesn't care
- Pull is a per branch property. You are pulling the specific branch that you are on

git pull

- Behind the scenes, "git pull" is a combination of
  - "git fetch" - pull data from remote server
  - "git merge" - merge the changes in that data
- All of the problems that can happen in a merge
- Added difficulty that now can be changes due to other developers

git push

- The opposite of pull
  - Pushes your changes to a code to the remote server
  - Will not generally work unless git can automatically merge those changes with the version on the server
    - “git pull” then “git push”
- Be careful! If not your repository people might not like you doing it
  - Shouldn’t be able to if you shouldn’t

git push

```
Marlow:demo bradyc$ git push
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 335 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/csbrady-warwick/DemoRepo.git
   b0f854a..bd04fad  master -> master
```

- If it works, should see something like that

- Push can be a much more complicated command if you want to push different local branches or the name of the local branch and the remote branch are different

- Read the documentation

Github

- GITHUB IS NOT GIT!
- By far the most popular public remote git server platform at the moment
- Easy to use
  - Gives a lot of help for setting up remote repositories
  - Same basic stuff that we've talked about here
- Provides a lot of nice extra features for developers
  - Support forums
  - Issue trackers

Github

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/csbrady-warwick/DemoRepo.git

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# DemoRepo" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/csbrady-warwick/DemoRepo.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/csbrady-warwick/DemoRepo.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

Forking

- Not part of git, but important when using GitHub collaboratively

- Part of GitHub flow model (https://guides.github.com/introduction/flow/)

- To contribute work to another person's repository you *fork* it

  - Make a copy of it that you control

Pull requests

- Also part of GitHub Flow
- If you've developed something on your forked copy but want your changes put back into the main repository
- Create a "pull request" asking the owner of the main repository to pull (as in "git pull") the changes from your repository

Conclusion

- Version control is very useful
  - Can save you a lot of time always
  - Can sometimes save your bacon
- Git is currently the most popular version control system
  - Bit daunting, but hopefully not too bad
- Don’t need to use all of it
  - Most developers are fine with init, add, checkout and commit