Git-ing Software and Document Version Control

Ethan Nelson

May 26, 2016

What is Git?

- Git is a version control system, or a software program that manages file changes.
- Git tracks how files change and provides the ability to view these changes at any point in time.
- Git also fosters collaborative development, allowing multiple people to update and maintain code repositories.
- Git is a very powerful tool and this session is only a very light brush of its features.

How does Git work?

- Git takes dissections of historical changes to deconstruct the changes in files from their present state.
- All of this occurs behind the scenes, and the results are stored in the .git/ directory of a repository.

Ease of examining differences in various file types

Type	Difficulty	Example extensions
Plain text	Easy	.TeX, .m, .f*, .pro, .py, .txt, .htm*
Rich text	Medium	.rtf
Proprietary	Hard	.docx, .pptx, .pdf

- *Any* file type can be tracked and versioned by Git, but the ease of looking at differences over time increases for non-plain text types.
- Additionally, for non-plain text types, Git often has to save the full file at every revision, so this adds up space-wise.
- These non-plain text types require an add-on (if one exists) to productively view changes.

Many use cases of Git

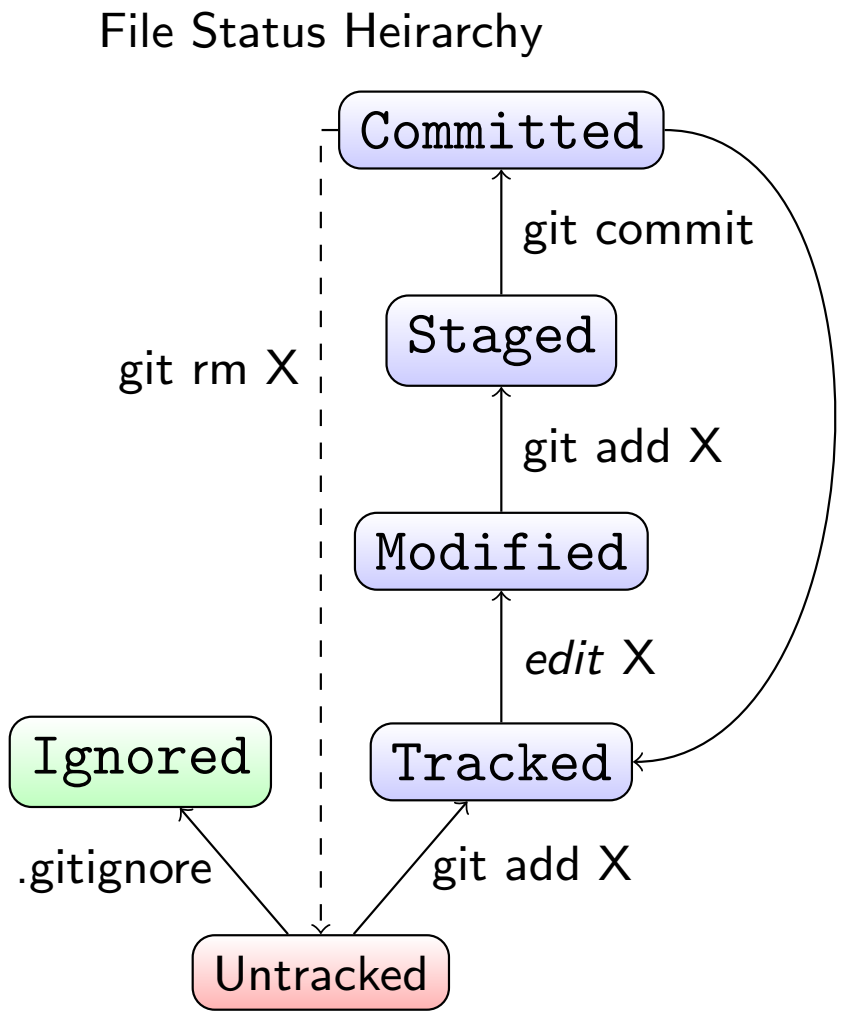
- Git has a few different cases or strategies in using it:
  - Simplest: you want to locally track changes to file(s) and subdirectories contained in a directory
  - A bit delicate: you want to locally separate development of code into a separate branch–e.g. a 'development' and 'live' version of code
  - A bit more delicate: you want to develop code locally and store changes on a master server
  - Even more delicate: you want to collaboratively develop code with others on a master server
- "Delicate" is used here over complexity: it is not necessarily hard–it just requires some learning and care.

Crash slide on Git vocabulary

- repository: the directory of code to be tracked (could contain one file, could contain 1000 subdirectories).
- commit: a version of files in the repository; like a snapshot in time.
- commit hash: a unique identifier for a commit.
- branch: a separate version of the repository that can be used to test changes (retaining all prior history).
- fork: a personal copy of someone else's repository (that includes the full history).
- push: merge your current changes into another fork or branch.
- pull: merge changes elsewhere into your current fork or branch.

File statuses within a Git repository

- Files within a Git repository can be of a few different states.
- Initially, all files are untracked.
- Files can be tracked, ignored, or continue untracked.
- Once tracked, files can be modified or unmodified since the last stage.
- Modified files can be staged to be committed.
- This is all on an individual file basis, meaning when you make a commit, you are able to save only changes to some files.



A little preconfiguration

- We need to add information about ourselves into Git first:

```
$ git config --global user.name "Bucky Badger"
$ git config --global user.email \
  "buckingham.m.badger@wisc.edu"
```

- This information is used by Git when creating a commit.

Now let's start with the basics!

- Open a terminal and create a directory for us to play in

```
$ mkdir temp-code; cd temp-code
```

- Initialize a Git repository in this directory

```
$ git init
Initialized empty Git repository in ./.git/
```

Now let's start with the basics!

- Check on the status of the repository

```
$ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add"
to track)
```

- We have no files in the directory, so there are none to be ignored, untracked, tracked, or staged.

- Let's say we start programming something

```
$ echo ''import numpy'' > script.py
```

- Now we have an untracked file in the directory

```
$ git status
#On branch master
#
#Initial commit
#
#Untracked files:
# (use "git add <file>..." to include in what will
be committed)
#
# script.py
#
nothing added to commit but untracked files present
(use "git add" to track)
```

- We can start tracking the file using the add command

```
$ git add script.py
```

- The status shows this as a staged file

```
$ git status
#On branch master
#
#Initial commit
#
#Changes to be committed:
# (use "git rm --cached <file>..." to unstage)
#
# new file:  script.py
```

- Now we want to commit the changes–preferably with a descriptive message

```
$ git commit -m 'Initial commit'
#[master (root-commit) a21bccb] Initial commit
# 1 file changed, 1 insertion(+)
# create mode 100644 script.py
```

- The status shows we have no changes to currently tracked files and also that there are no untracked files.

```
$ git status
# On branch master
nothing to commit, working directory clean
```

- The Git log includes our commit now

```
$ git log
commit a21bccbecbf3ea592d2be664b27b762e80c6c6ee
Author:  ethan-nelson <ethan-nelson@users.noreply.githu
Date:  Mon May 23 17:55:38 2016 -0500

Initial commit
```

- Assume we want to develop our code further and then look at the changes from the most recent commit

```
$ echo import scipy >> script.py
$ git diff script.py
diff --git a/script.py b/script.py
index c2e5936..2b86e99 100644
--- a/script.py
+++ b/script.py
@@ -1 +1,2 @@
import numpy
+import scipy
```

- You can use git diff without a file name to see the changes to all files in the repository.
- To reference prior commits, you can use git diff HEAD^^ HEAD, where you add as many carets as commits you would like to go back.

What about ignoring certain files?

- Sometimes you don't want to track files for one reason for another...
  - The dreaded .DS_Store on Macs
  - Temporary editor files (~, .swp)
  - Configuration files with private keys
  - Huge files that aren't very useful tracked (e.g. images)
- You can tell Git to ignore them by creating a .gitignore file in the top directory of the repository with files to ignore.
- This can be as general as *.~ to as specific as cats.jpg

- Let's create a .gitignore file in our directory now and add it to our repository

```
$ echo ''*.jpg'' > .gitignore
$ git add .gitignore
$ git commit -m 'Add gitignore'
[master (root-commit) b1d5296] Add gitignore
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 .gitignore
```

- To test the gitignore file, download a jpg file into the directory

```
$ curl http://ethan-nelson.me/cat.jpg -o cat.jpg
% Total % Received % Xferd Average Speed Time Time
Time Current
100 863k 100 863k 0 0 1801k 0 --:--:-- --:--:-- --:--:-
9.8M
```

- Finally, check the git status

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

- If you find a file type or subdirectory cannot be added, check the .gitignore file first!

Tagging commits

- If you want to actually version your code–i.e. assign it an incremental version as it is improved over time–you can use Git tags.
- The easiest tagging type is a lightweight tag, which simply assigns a tag at a certain commit.

```
$ git tag v0.1
$ git tag
v0.1
```

- You can view all tags or the corresponding commit

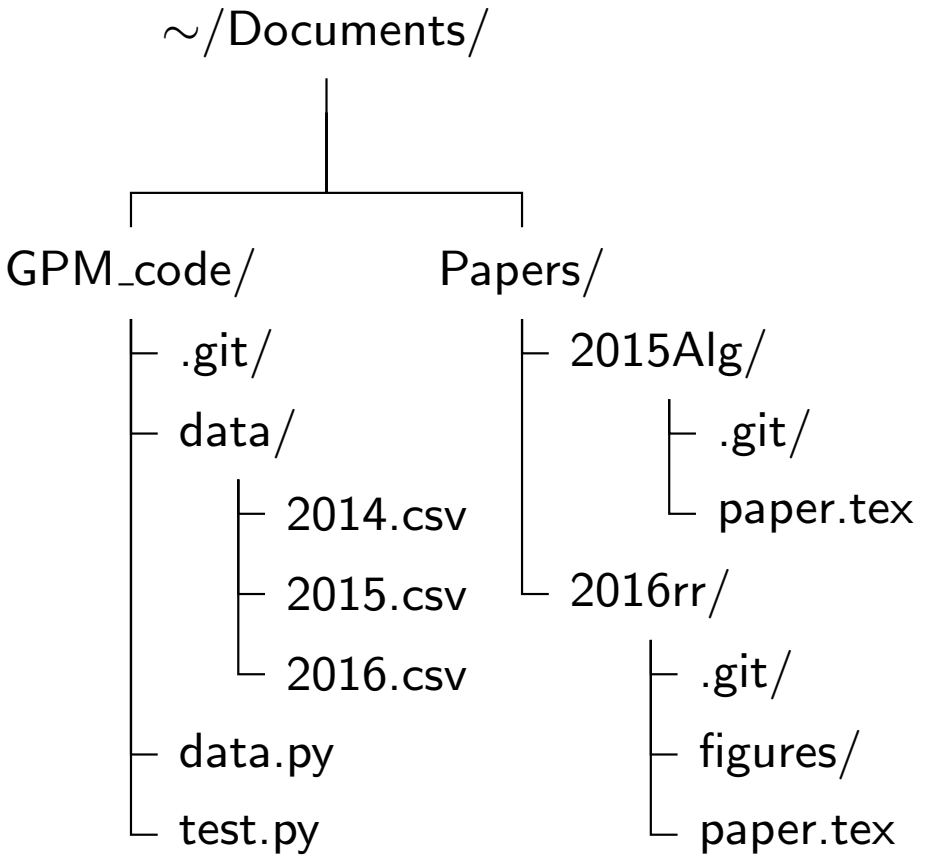
```
$ git show v0.1
commit b1d5296f73298607e68bf757db2f41c21cd86898
Author:  ethan-nelson <ethan-nelson@users.noreply.githu
Date:  Wed May 25 11:17:39 2016 -0500

Add gitignore

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..76ce7fc
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1 @@
+*.jpg
```

Concluding tips for this section

- You can initialize a git repository in any directory!
  - If you initialize the repository in an existing directory, all the files and subdirectories will default to untracked.
- Don't have git repositories within git repositories.
- Commit as often or as little as you like!

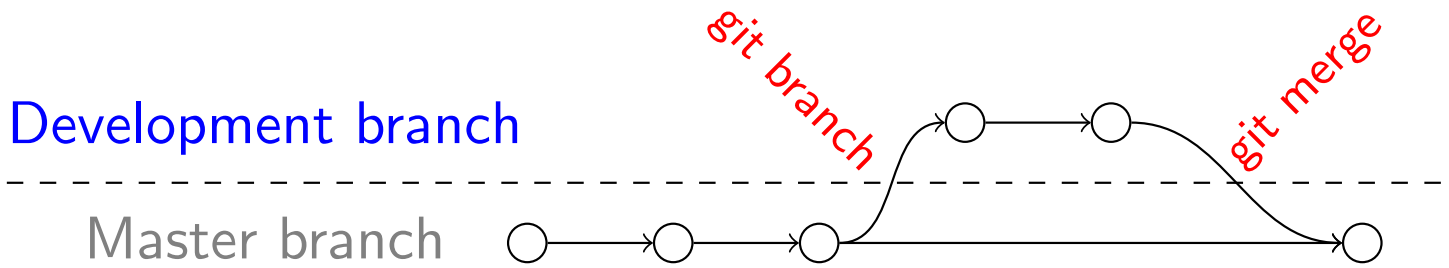


Working with two code branches

- What if the code you are working on should always have a functioning copy at a prior stage?
- Or what if you just want to make changes separately from your live code?
- This is where Git branches come in handy.
- Another branch is like a child of your "master" default code branch–or like a branch of a tree.
- The level of care required increases as we continue further...

Working with two code branches

- When you create another branch, that branch retains all of the previous history of the original branch.
- But all of the changes made in the secondary branch are not automatically applied to the original branch.
- To move changes back to your original branch, you use merge to combine them.



Working with two code branches

- Let's now create a branch of our code.

```
$ git branch
* master
$ git branch dev master
$ git checkout dev
Switched to branch 'dev'
$ git status
# On branch dev
nothing to commit (working directory clean)
```

Modifying code in a branch

```
$ echo "import sys" >> script.py
$ echo "IMPLICIT NONE" > script.f90
$ git status
# On branch dev
# Changes not staged for commit:
# (use "git add <file>..." to update what will be committed
# (use "git checkout -- <file>..." to discard changes in
working directory)
#
# modified:  script.py
#
# Untracked files:
# (use "git add <file>..." to include in what will be commi
#
# script.f90
```

Committing to a branch

- Now let's add our modifications and new files, then commit.

```
$ git add -A
$ git commit -m 'Introduce fortran'
[dev 516713a] Introduce fortran
2 files changed, 2 insertions(+), 0 deletions(-)
create mode 100644 script.f90
```

Comparing branch files

- We can use the git diff command to compare branches

```
$ git diff dev master
diff --git a/script.f90 b/script.f90
deleted file mode 100644
index d221114..0000000
--- a/script.f90
+++ /dev/null
@@ -1 +0,0 @@
-IMPLICIT NONE
diff --git a/script.py b/script.py
index 591df7e..345e6ae 100644
--- a/script.py
+++ b/script.py
@@ -1,2 +1 @@
import scipy
```

Comparing branch commits

- It's not as easy to compare branch commits as it is file differences.
- My recommendation is

```
$ git log master..dev
commit 516713a1e331e1dbc24a780dbf5b002c43e58861
Author:  ethan-nelson <ethan-nelson@users.noreply.githu
Date:  Wed May 25 15:09:11 2016 -0500

Introduce fortran
```

- This shows the additional commit in the dev branch compared to master.

Merging branches

- Assuming we are satisfied with our dev changes, we can merge them back into our master branch.

```
$ git checkout master
$ git merge dev
Fast-forwarding to:  dev
Merge made by octopus.
script.f90 1 +
script.py 1 +
2 files changed, 2 insertions(+), 0 deletions(-)
create mode 100644 script.f90
```

Pushing commits

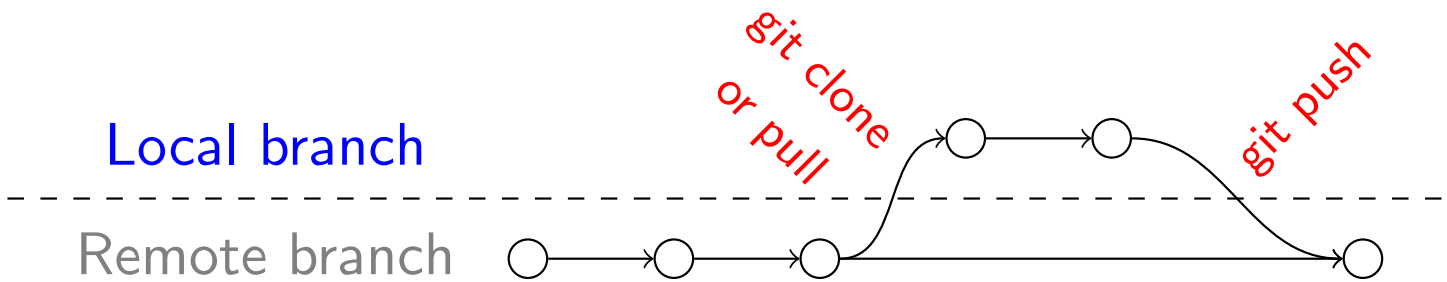
- We're already in the more advanced branch (dev), so we want to push our changes to the less advanced branch (master).

```
$ git log master..dev
commit 516713a1e331e1dbc24a780dbf5b002c43e58861
Author:  ethan-nelson <ethan-nelson@users.noreply.githu
Date:  Wed May 25 15:09:11 2016 -0500

Introduce fortran
```

Working with a remote code repository

- From here on, features are in line with someone who is collaboratively developing with others or otherwise using a server as their main codebase.
- This server could be a file server in the department, GitHub, GitLab, or something else.
- Git's flexibility with distributed version control becomes very useful here.



- Lets assume you want to work on development of a code base from GitHub.
- We will clone the code repository as is from GitHub.

```
$ cd ../; mkdir temp_code2; cd temp_code2
$ git clone https://github.com/ethan-nelson/code.git
Cloning into directory code...
remote:  Counting objects:  17, done.
remote:  Compressing objects:  100% (13/13), done.
remote:  Total 17 (delta 2), reused 0 (delta 0), pack-r
Unpacking objects:  100% (17/17), done.
```

- The code repository from the GitHub server has now been copied–with complete history–in the code directory.

```
$ cd code
$ git status
# On branch master
nothing to commit (working directory clean)
$ git log
commit 9239efc3c34e18aaf4611b0cba677266b90ad07c
Author:  Ethan Nelson <ethan-nelson@users.noreply.githu
Date:  Thu May 26 09:15:50 2016 -0500

Add gitignore file

commit aa75748aa712fef8e208e2bb7b9c9fcad8e1650d
Author:  Ethan Nelson <ethan-nelson@users.noreply.githu
Date:  Thu May 26 09:14:55 2016 -0500
```

- Now let's assume time has passed and someone has updated the remote repository with more commits and you haven't changed your code yet. To sync your local repository, you can *pull* the changes from the server.

```
$ git pull origin master
remote:  Counting objects:  3, done.
remote:  Compressing objects:  100% (3/3), done.
remote:  Total 3 (delta 1), reused 0 (delta 0), pack-re
Unpacking objects:  100% (3/3), done.
From https://github.com/ethan-nelson/code
* branch master -> FETCH_HEAD
Updating 9239efc..0e9c605
Fast-forward
__init__.py 1 +
1 files changed, 1 insertions(+), 0 deletions(-)
$ git log
...
```

- The log now shows the updated commit
- To update the repository with your local changes, you can *push* your changes to that server.

```
$ git push origin master
(This will actually fail since you don't have pushing
privileges to my GitHub.)
```

Dangerous commands

- If you are using more than one branch and try merging changes from different branches with different histories, conflicts will likely arise.
- Git has capabilities built in for resolving these but I'm not going to cover them, so read up online before doing so :)
- Using commands like git checkout and git reset can erase any changes since your last commit. NEVER USE THEM EXCEPT IN VERY EXTREME CASES.
- Git is not a substitute for backups!

Staying out of the red

- If you follow the cycle of git add, git commit, git add, git commit, you should never run into any trouble.
- When using multiple branches, always ensure you are on the right branch before making edits.
- Going back in time (e.g. undoing parts of commits) is always possible; recovering uncommitted and lost changes is not!
- When in doubt, add, commit, and fix from there.

How to get Git

- Download and install like any other program.
  - Any operating system that is not OS X below Lion: git-scm.com/download
  - OS X below Lion: git v1.8.4.2 is recommended
- Graphical interfaces exist, too: git-scm.com/downloads/guis

GitHub

- Github is a free host for public-facing git repositories (though only users given permission can push commits).
- Content hosted ranges from simple programs or to-do lists to complex software applications.
- Github is a centralized repository that provides a web interface to browse code, examine commit history, and edit code.
- Github also has features that include issue tracking for bugs and feature requests, wikis describing the repository, and even web hosting.

GitHub

- Paid plans with Github provide private repositories that can only be accessed by authorized users.
  - Students can get free micro pro accounts and educators/researchers can get discounted service (possibly free), too: education.github.com

Thanks for listening! Questions?

Docs: git-scm.com/book