Git 101

or

"How I Learned to Stop Worrying and Love the Git"

Adam Hayes

National Nuclear Data Center
Brookhaven National Laboratory

05.2020

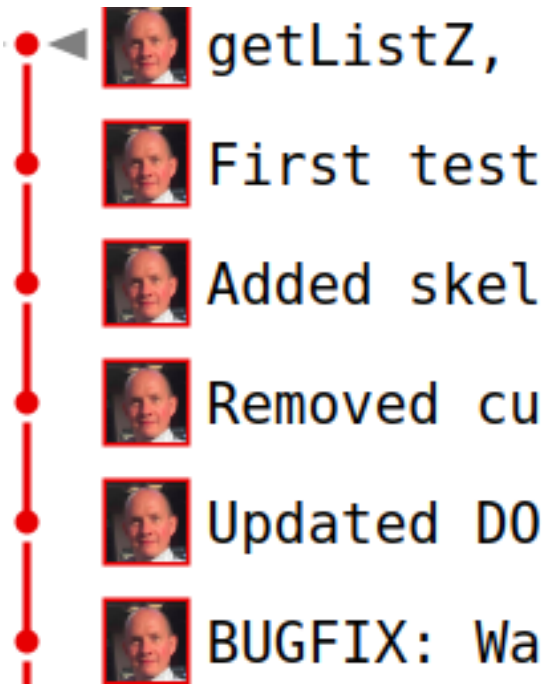
What is Git?

- Version control / "Revision control"
- Linus Torvalds, 2005
- *noun* A contemptible person
- Alternative to Subversion, Mercurial, etc.
- Does not "contain" your files; you are not bound to Git
- Tracks *content*, not files
- Work locally; remote server is *optional*
- **Git** $\neq$ **Gitlab**

git-scm.org

Why Git / Why version control?

- Save all versions
- Separate changes by topic
- Simplify debugging
- Facilitate collaboration
- Avert disaster–item Peace of mind
- Most appropriate for text
  - code
  - LaTeX docs (this slideshow)
- Works fine for binaries
  - Word docs
  - PPT



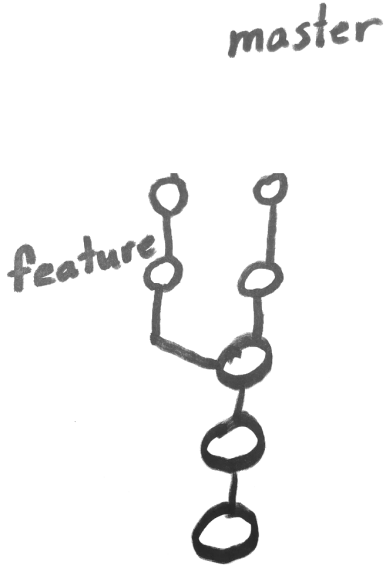
Suggestions for today

- Don't try to remember *how* to do everything
- Use these slides (link at end)
- Search online for details

We'll see short overviews and do some demos after each.

Terminology

- Commit: a saved version
- History: all of the commits
- Branch: (by illustration) main branch is "master"
- HEAD: the most recent commit
- Merge: automatically merge changes from one branch into another
- Repository ("repo"): a project managed using Git



Creating a repo

1. In the project directory (top level), type
   `git init`

That's it!

Making commits

- Add the files you want to commit:
  `git add <file name>`
- Make the commit:
  `git commit`
- Type a detailed commit message for others (~~"Fixed bug"~~)

That's it.
See commit hash in demo.
Note: best not to commit executables or "generated file" e.g. PDFs from Latex output

Commit messages

- Short lines
- Second line blank
- Add detail (important for debugging)
- Git generates a "hash"

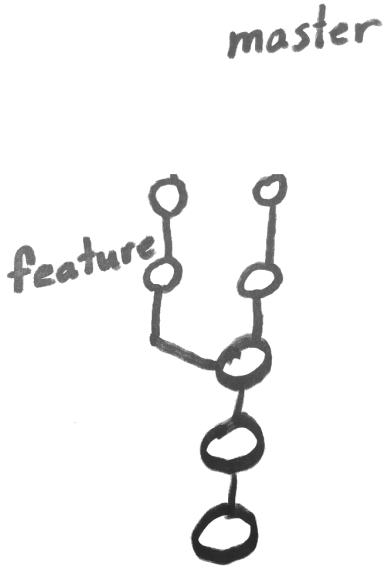
```
commit 8dbbafa1631b7684dcc5cf6770cfdc93bef29e4a
Author: Adam Hayes <ahayes@bnl.gov>
Date:   Mon Apr 20 21:16:49 2020 -0400

    BUGFIX: Was not giving error when trying to overwrite a document

    webapp.cgi now returns DocumentExists error response when the client
    tries to insert a document with an _id that already exists.
```

Branching

- Create a branch and check it out
  "`git checkout -b feature`"
- Branch need not be pushed to server
- No one will know unless you push
- Can jump between branches at will
- Branches are cheap



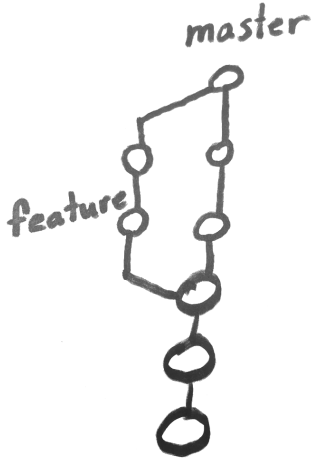
Checking out

- Jump to branch
  "git checkout <branch name>"
  e.g.
  "git checkout add-ensdf-script"
- Jump to commit
  "git checkout <commit hash>"
  e.g.
  "git checkout c010a24"
- Return to master
  "git checkout master"

```
* 05410a7 (HEAD -> get-nuc-lists, origin/get-nuc-l
* 577163d (origin/master, master) Removed curl exa
* c010a24 Updated DOCS.md for DELETE request and c
* 8dbbafa BUGFIX: Was not giving error when trying
| * 8907e88 (origin/add-ensdf-script, add-ensdf-sc
| * 5ef24f6 add-ensdf-to-wc.py: changed keys to re
| * 6d4eca5 add-ensdf-to-wc.py: Turned off some de
| * ea252df Updated DOCS info on DELETE requests
| * f1364d4 add-ensdf-to-wc.py: Fixed un-escaped u
| * 3369e12 add-ensdf-to-wc.py: Took % symbols out
| * 9467d19 add-ensdf-to-wc.py: Chg denergy to []
| * 7bbbf60 add-ensdf-to-wc.py: Corrected 'symbol'
| * 416d7dc add-ensdf-to-wc.py: can insert new doc
| * dd61e49 add-ensdf-to-wc.py: Can check for exis
| * dec681f add-ensdf-to-wc.py: Translating Jpi co
| * e872bd5 add-ensdf-to-wc.py: Changed energy uni
| * 8ac64e0 add-ensdf-to-wc.py: removed debug prin
| * 6ca4e50 add-ensdf-to-wc.py: decay mode handles
| * 0ea3108 add-ensdf-to-wc.py: limited stored sta
```

Merging

- Git automates this pretty well
- Fix conflicts marked by Git right in your text file(s)
- Syntax:
  `git merge mybranchname`"
- Think "Merges the branch *you named* into the branch *you're on*"



The (optional) server: "Upstream"

- e.g. Gitlab, Github...
- Clone existing repository: `git clone <url>/<myreponame>`
- Push/pull to/from remote server
  `git push`
  `git pull`
- Usually called "origin"

Creating a repo

1. Create first on Gitlab
2. Follow instructions to clone
   or
   push your existing repo

(Quick demo in browser)

Group projects: best workflow

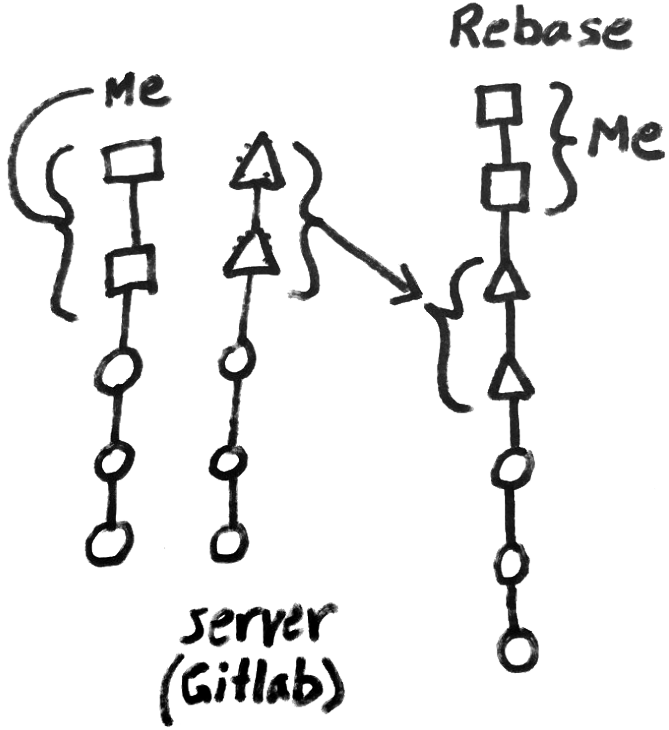
1. `git pull` (Make sure you have the latest commits)
2. Work
3. Test!
4. `git add ...`
5. `git commit`
6. `git push` (right away, please)

Demo...

What if...?

What if someone else made changes before I pushed?

- Don't panic; it's fine.
- `git pull --rebase`
- Resolve conflicts if necessary
- `git push`



.gitignore file

- Add the files / file types that you *never* want to commit
- Git won't bug you about them in "`git status`"
- You won't commit them by accident
- Probably every project will have a .gitignore file

```
# Prerequisites
*.d

# Compiled Object files
*.slo
*.lo
*.o
*.obj

# Precompiled Headers
*.gch
*.pch

# Compiled Dynamic libraries
*.so
*.dylib
*.dll
```

Fancy things...

Don't try to remember *how* to do this stuff.

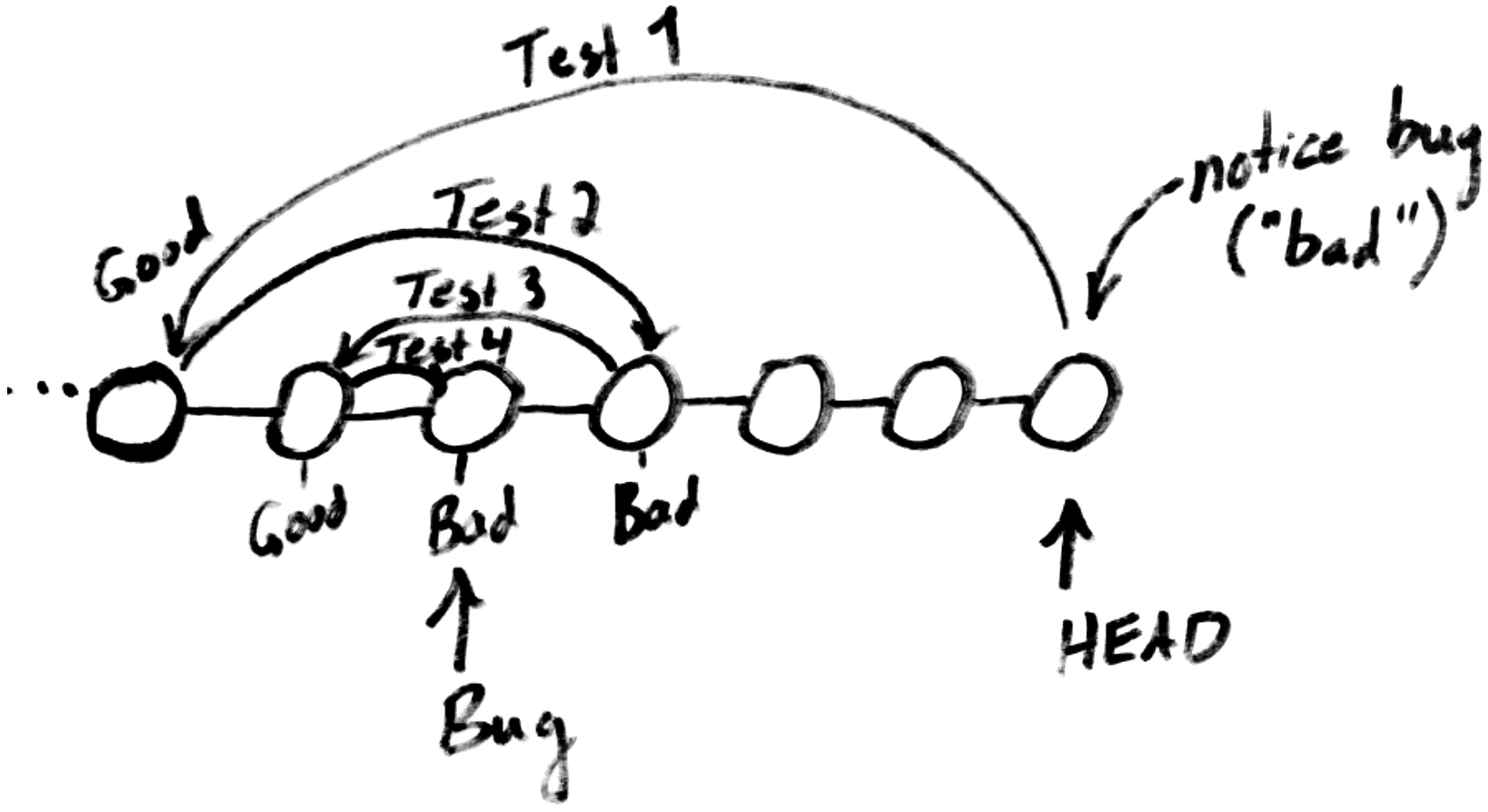
Tagging special versions

- `git tag <version>`
- Other options
- e.g. `git tag v1.0`
- `git push --tags`
- Tags are completely optional

Debugging with Git: Bisect

- Does a binary search for bugs
- Google "`git bisect`"
- Find the first commit with the bug
- "`git diff`" to help find the cause
- **Procedure can be automated with a test script!**

Debugging with Git: Bisect



Stashing

- Google "`git stash`"
- Stash work in progress ("WIP")
- Useful for jumping to another branch before you're ready to commit

Blame

```
git blame <file>
```

```
30176d0d (Lukas Flling 2014-02-07 12:18:20 +0100 227)  if(height<=0){
30176d0d (Lukas Flling 2014-02-07 12:18:20 +0100 228)          if(speed>10){
30176d0d (Lukas Flling 2014-02-07 12:18:20 +0100 229)                  printf("%s", dead);
30176d0d (Lukas Flling 2014-02-07 12:18:20 +0100 230)          }
20f4f72f (k3ll4gr0up    2014-03-30 22:49:00 +0200 231)          else if(speed<10 && speed>3){
30176d0d (Lukas Flling 2014-02-07 12:18:20 +0100 232)                  printf("%s", crashed);
30176d0d (Lukas Flling 2014-02-07 12:18:20 +0100 233)          }
20f4f72f (k3ll4gr0up    2014-03-30 22:49:00 +0200 234)          else if(speed<3){
30176d0d (Lukas Flling 2014-02-07 12:18:20 +0100 235)                  printf("%s", success);
30176d0d (Lukas Flling 2014-02-07 12:18:20 +0100 236)          }
```

Recommendations and resources

- Work on the "command line;" avoid the GUIs
- Use the best workflow:
  1. `git pull` (Make sure you have the latest commits)
  2. Work
  3. Test!
  4. `git add ...`
  5. `git commit`
  6. `git push` (right away, please)
- Make *small, topical* commits
- Don't do `git add *`; don't commit these:
  - "generated output" e.g. PDFs from Latex
  - gigantic files
  - Passwords or other sensitive info; history is forever
- Work on side branch, test, merge; "master always works"

Recommendations and resources

- Git is easy; learn the fancy things as you need them.
- Remember the **things you can do**, Google for syntax
- Resources
  - Google, stackoverflow.com
  - Official site for details: `https://git-scm.com`
  - Install on Ubuntu: `sudo apt-get install git`
  - Install on Mac: `https://git-scm.com/download/mac`
  - Install on Windows: `https://git-scm.com/download/windows`
  - Some come with UI. Please don't use it. You'll make mistakes.
  - Make alias
    `git log --oneline --graph --decorate --branches`
    (ask Adam)

Quick recap demo

If you remember *nothing* else...

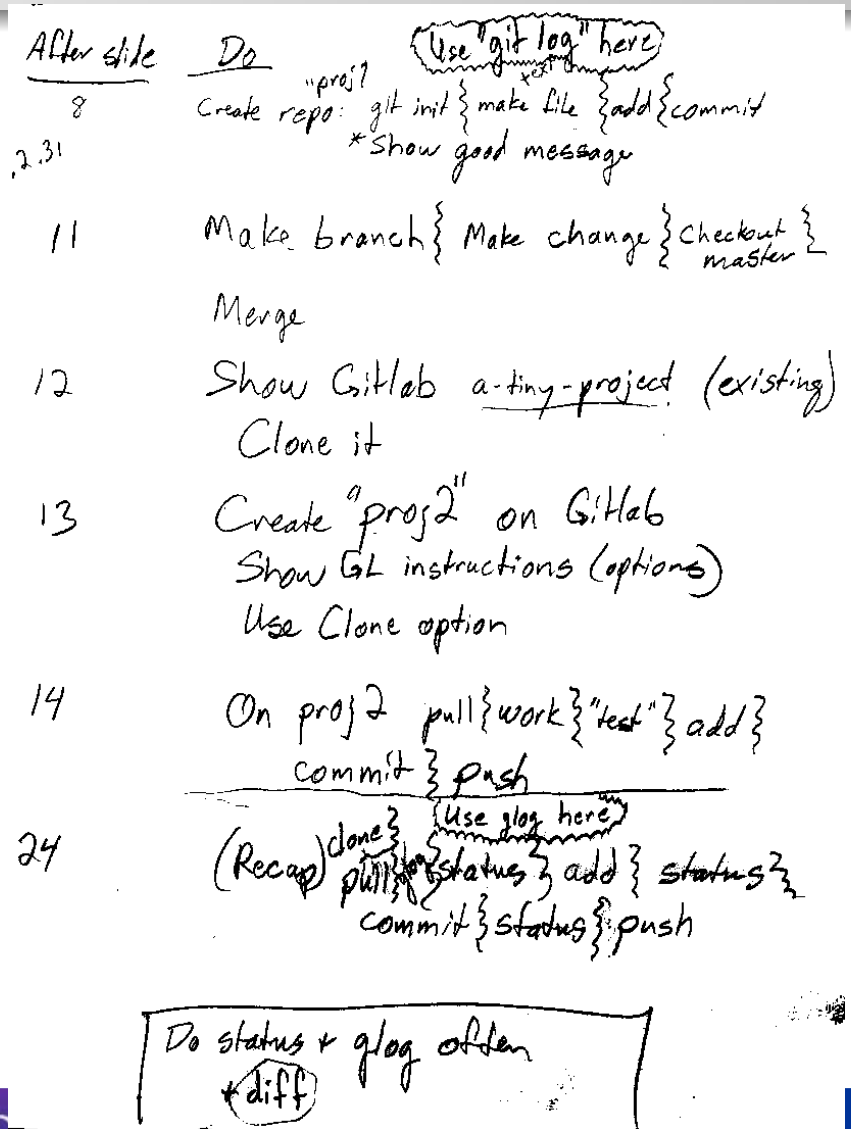
- Clone "a-tiny-project" from Gitlab
- Check status and log
- Add a file
- Check status
- Commit a file

- These slides can be found here (soon):
  `https://development2.nndc.bnl.gov/internaldocs/git/`

End

END

Notes


After slide
Do
Use "git log" here
8
Create repo: "proj" git init { make file { add { commit
* Show good message
11
Make branch { Make change { Checkout master {
Merge
12
Show Gitlab a-tiny-project (existing)
Clone it
13
Create "proj2" on Gitlab
Show GL instructions (options)
Use Clone option
14
On proj2 pull { work { "test" { add {
commit { push
24
(Recap) clone { Use glog here
pull { status { add { status {
commit { status { push
Do status + glog often
* diff