PHYSICS & ASTRONOMY
TEXAS A&M UNIVERSITY

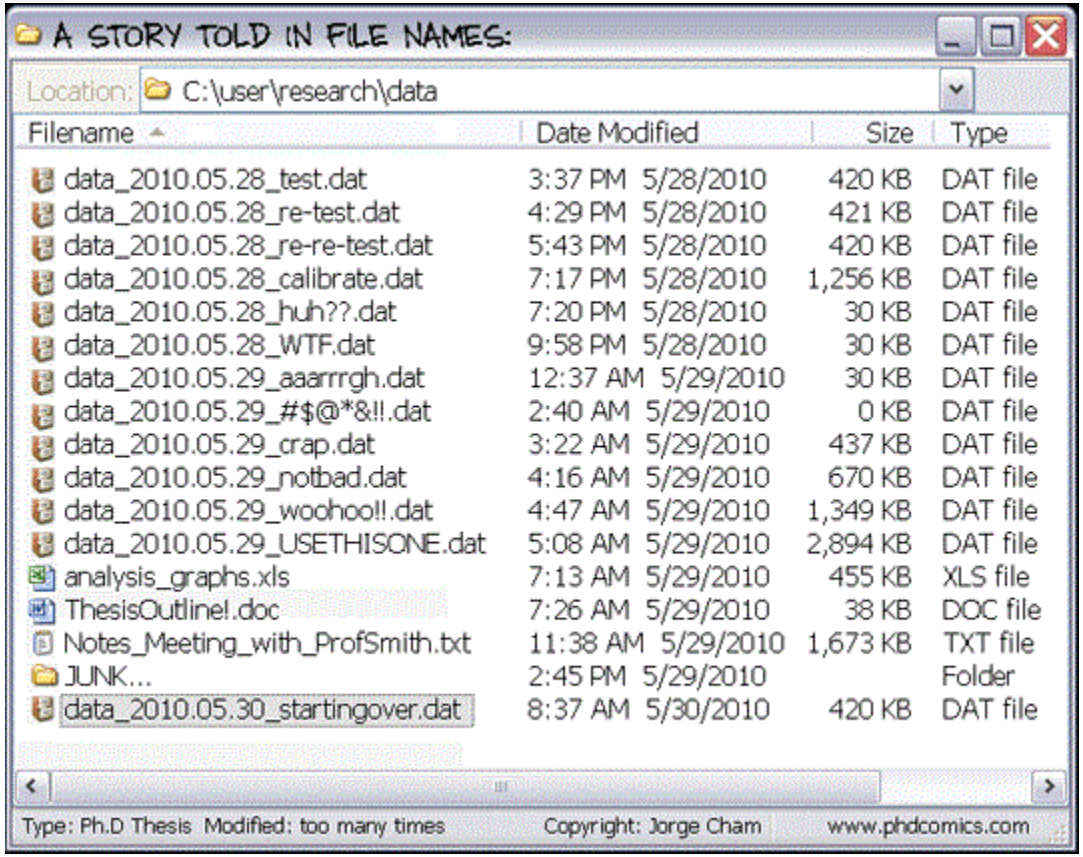
Introduction to Git

James Gerity

August 12, 2016

Slides available at http://www.github.com/jgerity/talks

Is this familiar?



"Piled Higher and Deeper" by Jorge Cham
www.phdcomics.com

Problems with this approach

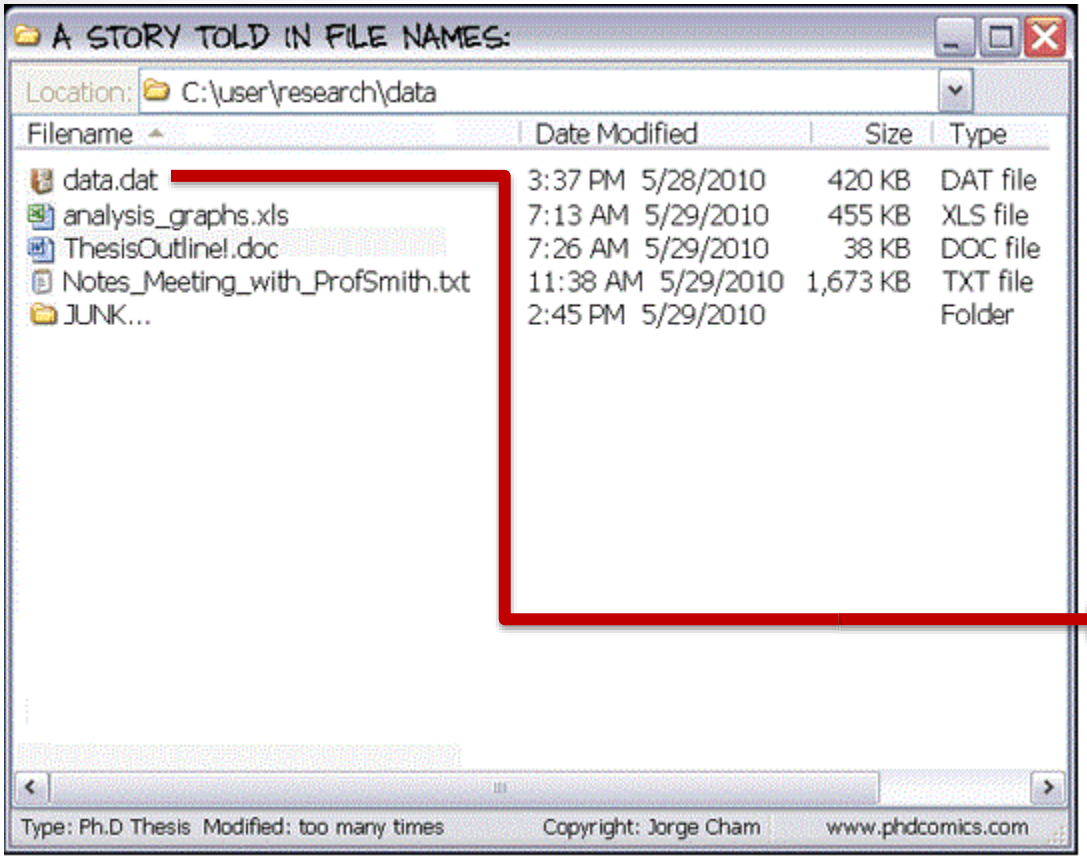
Version history

- Reproducible research
- Screwing up is *part of the process*, but we're afraid to destroy/lose work we've already done!

Collaboration

- Everyone's crazy systems do not work well together
- Gets out of sync very quickly
- Can't tell what each person did without laborious manual documentation

A better way… Version Control



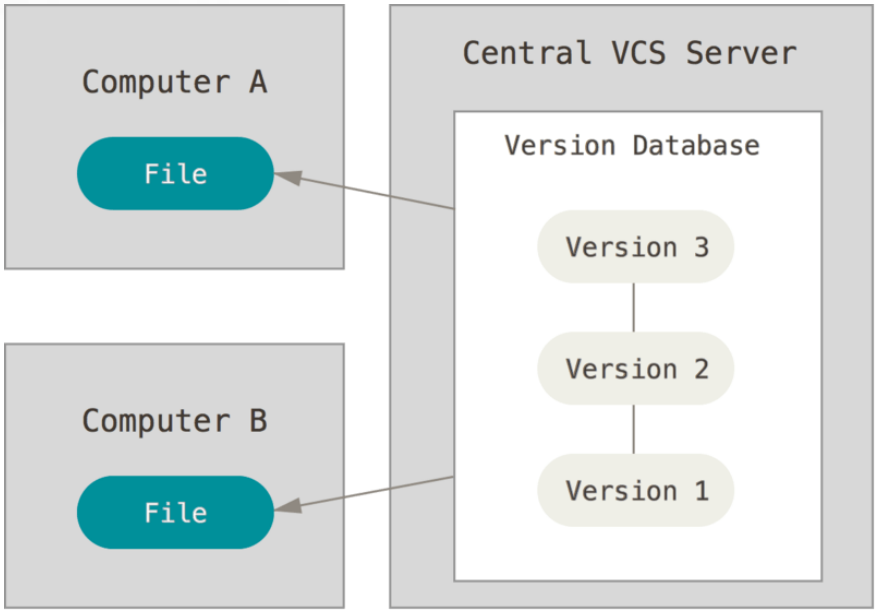
"Piled Higher and Deeper" by Jorge Cham
www.phdcomics.com
(image edited)

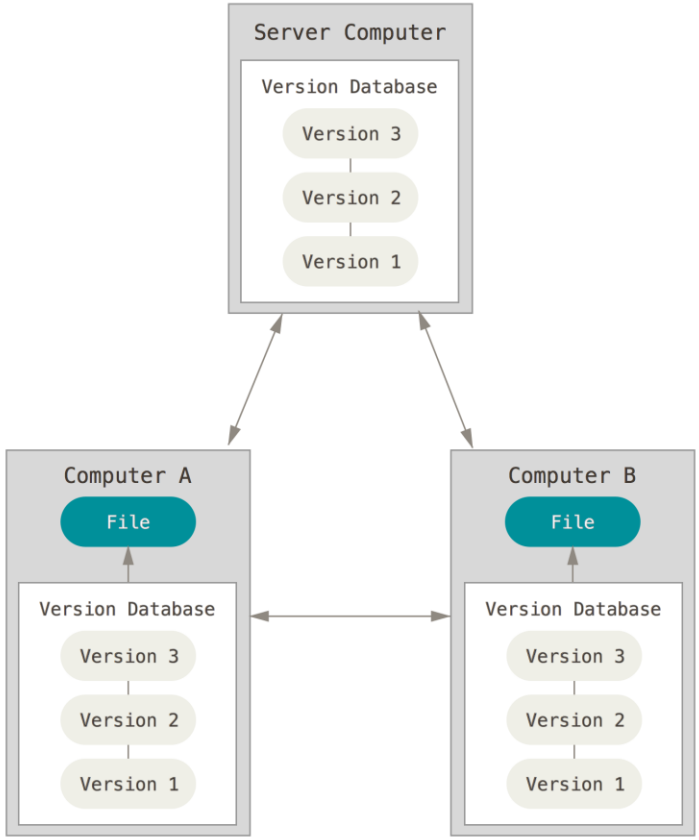
OTHER VERSIONS		
Message	Date	Size
Changed the setup	4:29 PM 5/28/2010	421 KB
Second try	5:43 PM 5/28/2010	420 KB
Calibrated the thing	7:17 PM 5/28/2010	1,256 KB

What is Git?

- Git is distributed version control software.
- Version control means it handles file versions
- Distributed means everyone has a full copy of the repository, and all the information therein.
  - There is no ‘central’ database, by design.

Centralized vs distributed model



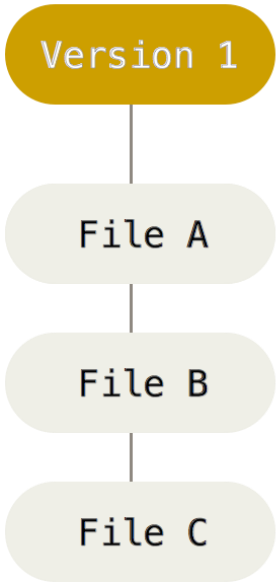


The scope of this talk

- The core concept of Git's data model
- How to do some basic operations in Git
  - I'll briefly touch on more advanced topics like *branching*, *merging*, and working with *remotes*, but it's beyond an introductory scope
  - Feel free to ask me some questions later!

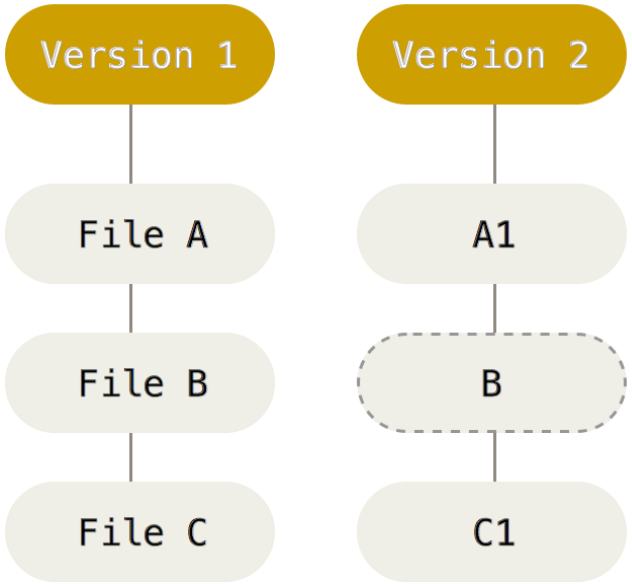
How Git works, the short version

- Let's start at version 1 with a set of files



How Git works, the short version

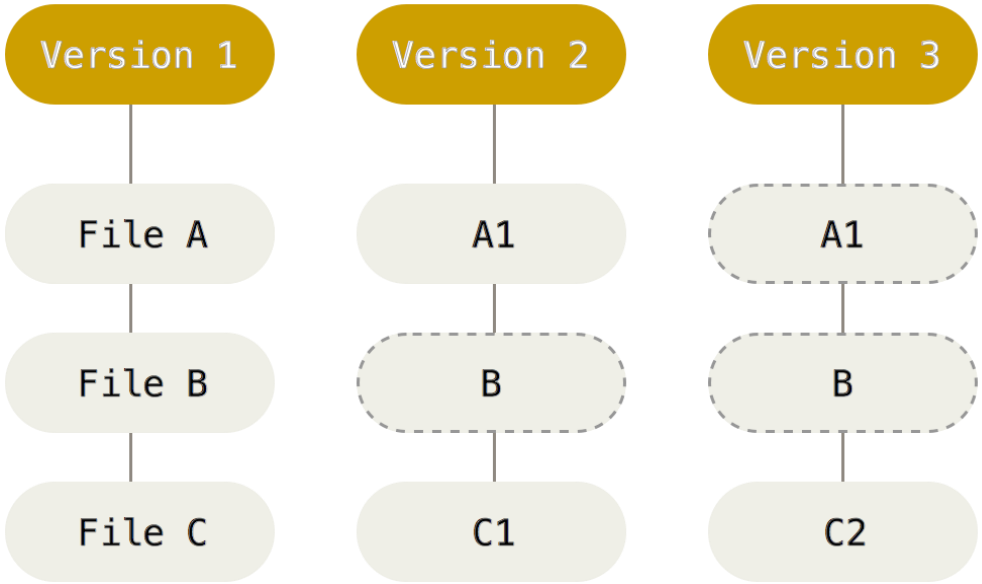
- We change files A and C and create a new version



- File B didn’t change, so point to the old version of it.

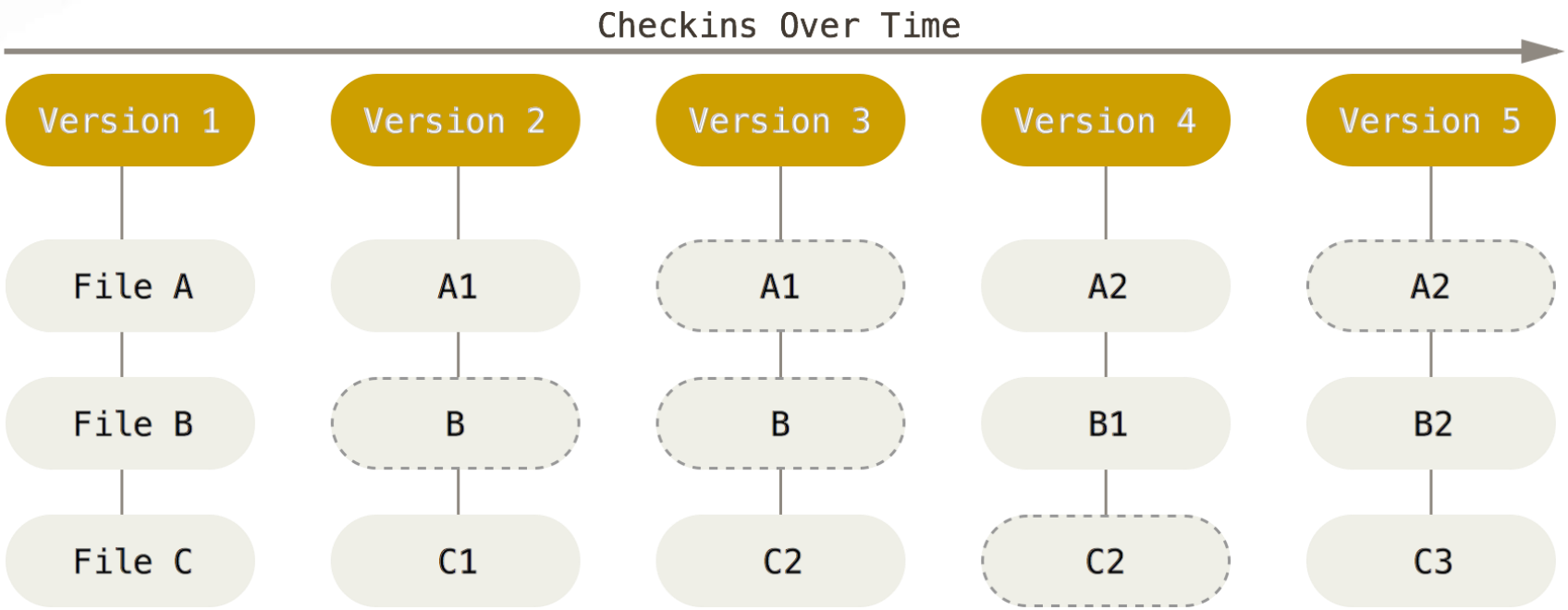
How Git works, the short version

- We change C again, but not file A. We'll keep pointing to version A1 of it.

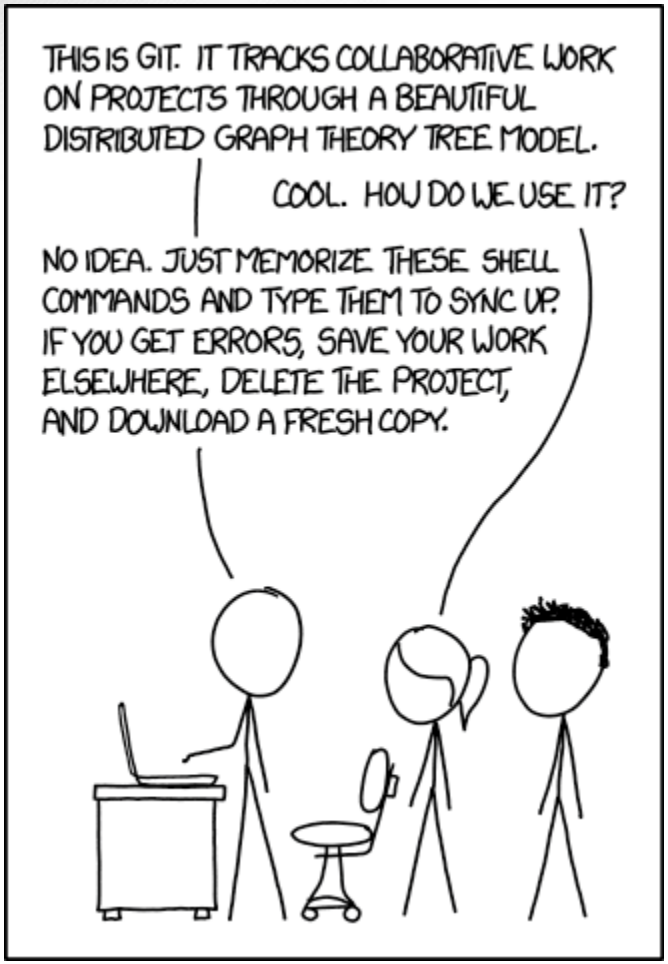


How Git works, the short version

- Ad nauseum



Is this familiar?



- Tools are only as good as our ability to use them effectively!
  - Sometimes, Git is overkill
  - When it’s not…

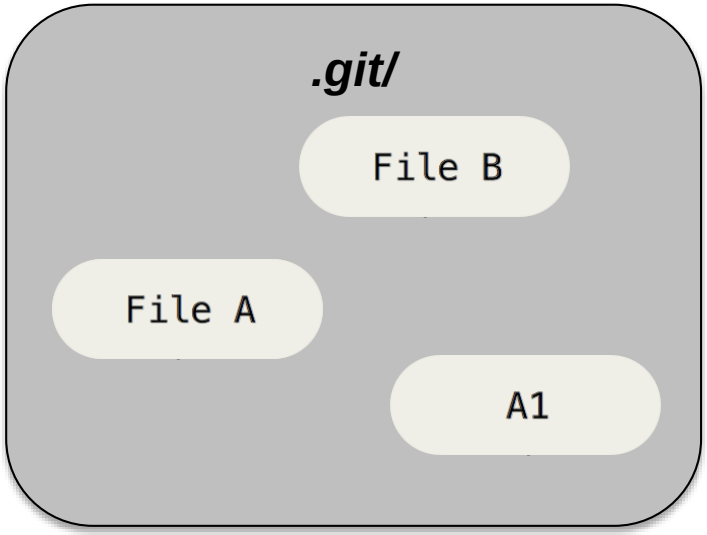
How Git works, the less-short version

- To fully understand how Git works, you need to understand 3 types of objects:
  - Blobs (file data)
  - Trees (directory layout)
  - Commits (trees + metadata)

What's under the hood? - blobs

- At the most basic level, Git stores literal copies of files it is aware of. These are called *blobs*, and are stored in the .git folder.

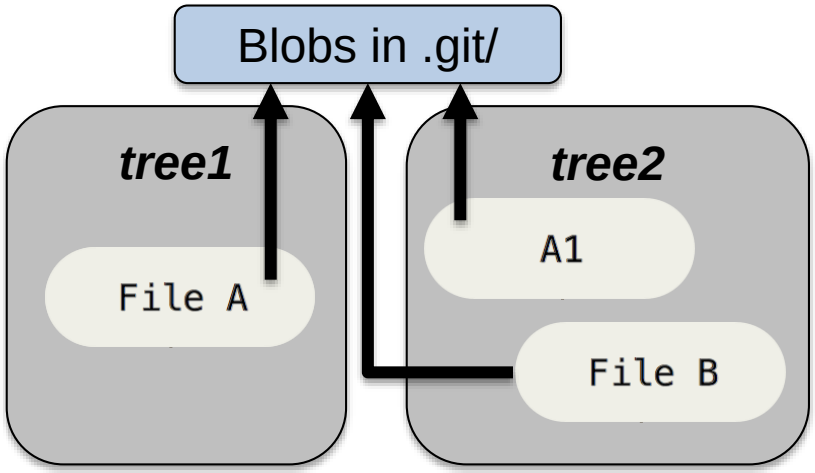
- If a file doesn't change, we don't need a new copy!



What's under the hood? - trees

- To point to different blobs and describe how they're organized, we use *trees*. A tree is just a list of blobs and **other trees**. Think of them as directories.

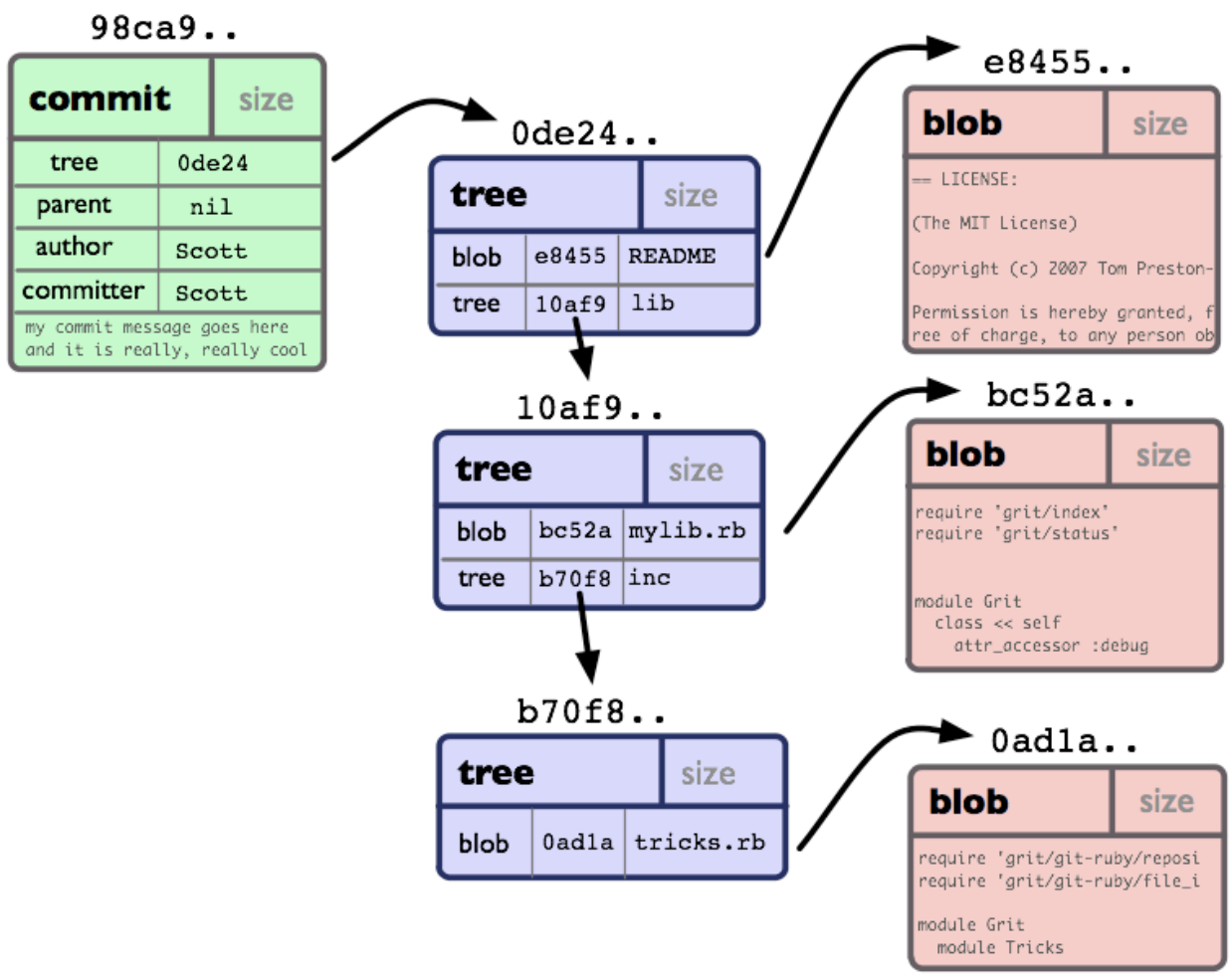
- Key point: trees do not *contain* blobs, they just point at them.



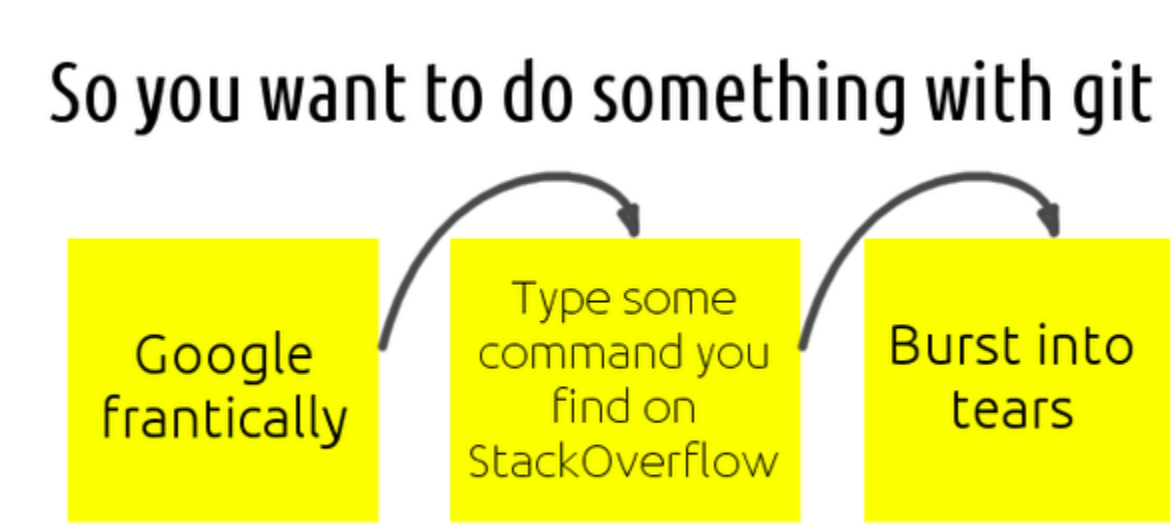
What's under the hood? - commits

- A *commit* is the thing you'll work with the most, but it's just a special way to talk about a tree. A commit is a tree + metadata about that tree (what came before it, who authored the commit, etc.)

What's under the hood? - commits

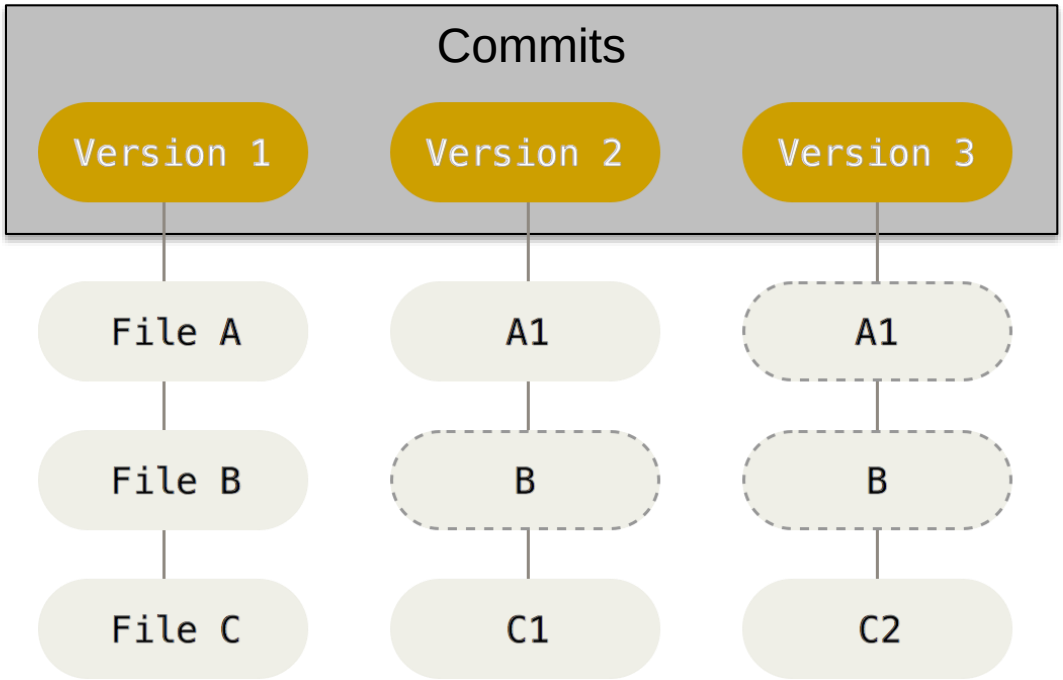


OK, so how do I actually do stuff in Git?



OK, so how do I actually do stuff in Git?

- Always remember: the basic "unit" of a Git workflow is the commit

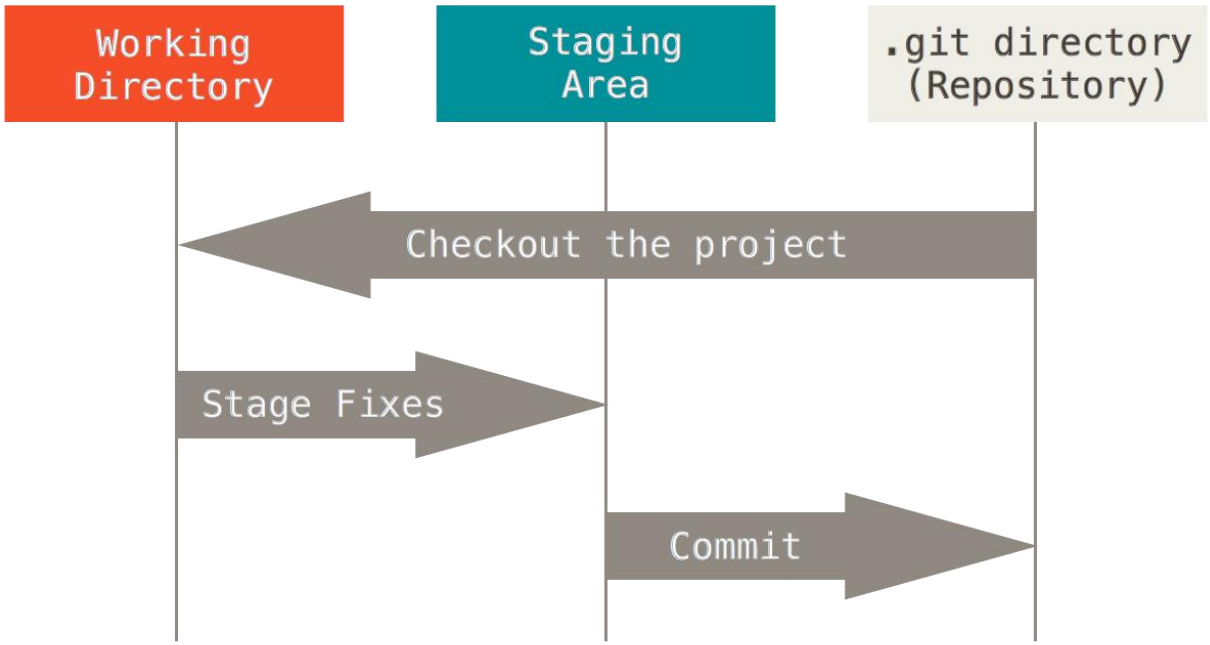


OK, so how do I actually do stuff in Git?

- When working with Git, commits are typically built in 3 steps:
  1. Edit files (write code, etc.)
  2. Tell git which files have changed
  3. Tell git to save changes as a commit

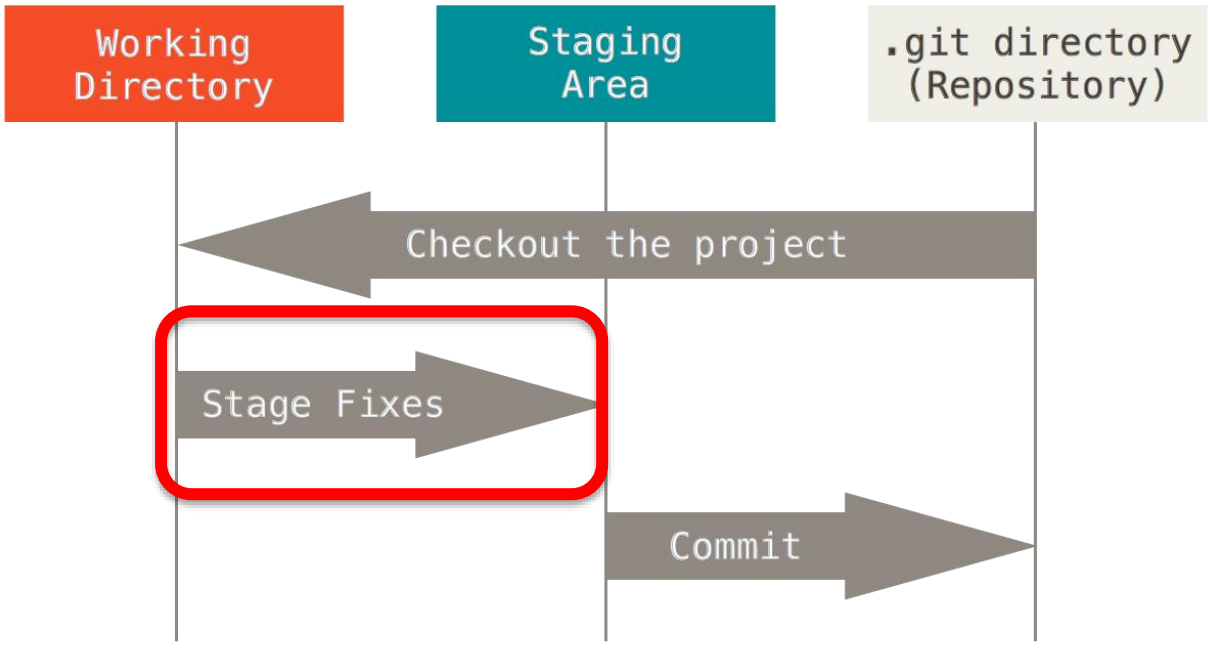
OK, so how do I actually do stuff in Git?

- When working with Git, commits are typically built in 3 steps:



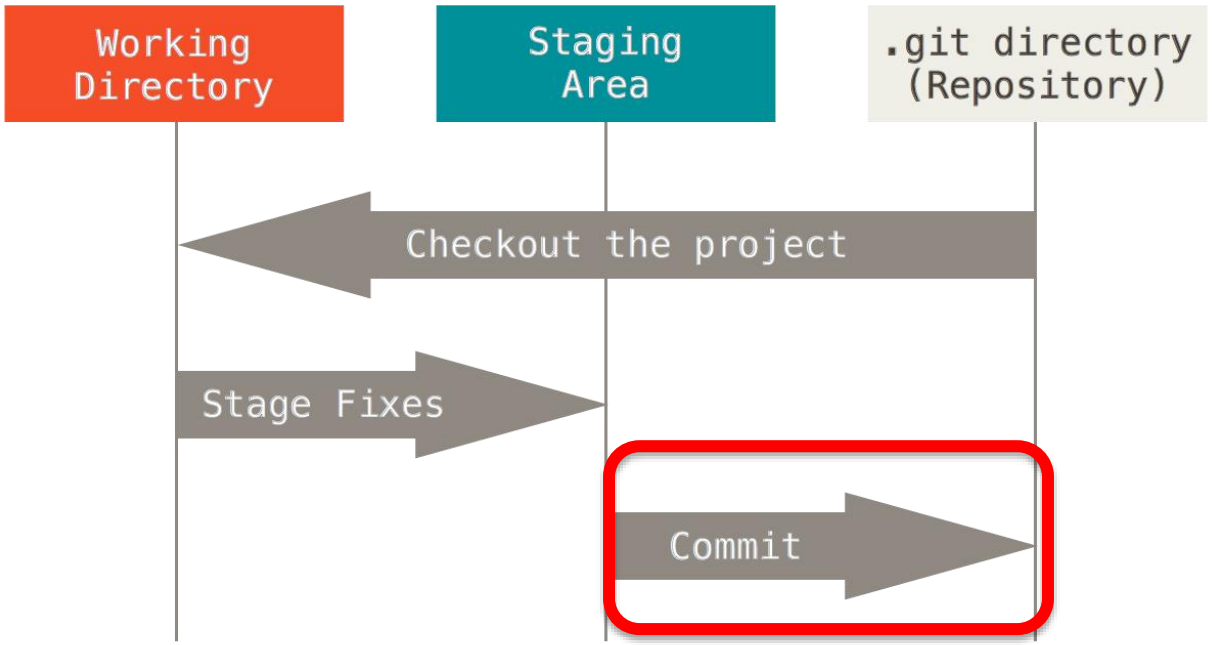
OK, so how do I actually do stuff in Git?

- Running the git add filename command tells Git to ‘stage’ the specified file/directory



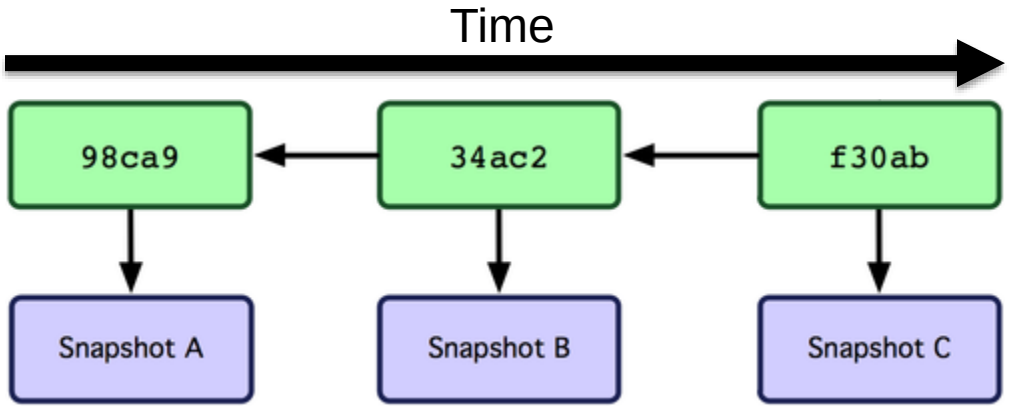
OK, so how do I actually do stuff in Git?

- Running the git commit command tells Git to compile all staged items into a commit



Commit history

- Once you create a commit, it is stored in the database, and is likely the “newest version” of your repository.
- Don’t forget, though, that Git’s database remembers what old commits looked like!



What is GitHub?

- Since Git is distributed, in order to share your work with others, you must distribute your repository to them.
  - …and for anyone to share their work with you, vice versa!
- Typically, we want to store a repository somewhere to centralize all this *pushing* and *pulling* of repository data.

What is GitHub?

- GitHub.com is just a free website that facilitates this exchange process
- You, as the owner of a repository can make local changes and *push* them to GitHub (update the remote copy)
- If I want to contribute to your work, I copy (*fork*) your repository, change it, and then request that you *pull* data from my repo

A tour of branches in Git

- A *branch* is an object that points to a particular commit. There is always at least one branch, usually named “master.”
- If we created a new commit, we’d move master ‘forward.’

Figure from http://hades.github.io/2010/01/git-your-friend-not-foe-vol-2-branches/

A tour of branches in Git

- Here, we have a second branch called ‘nice_feature’ that tracks another set of commits.
- Since the two branches diverge, we can work on one without changing the other, which is often beneficial to development.

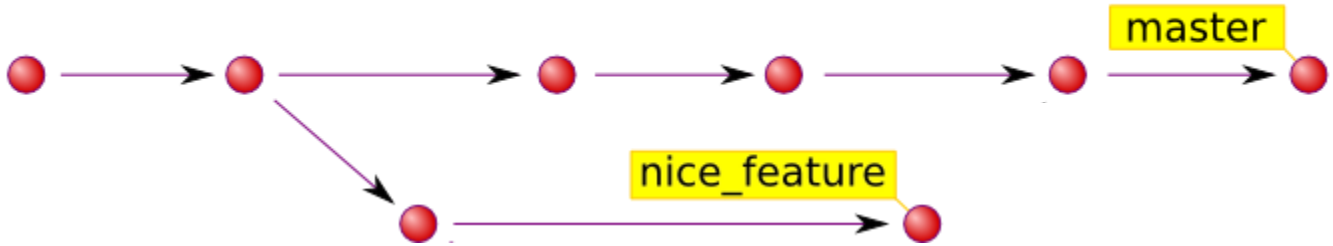


Figure from http://hades.github.io/2010/01/git-your-friend-not-foe-vol-2-branches/

A tour of branches in Git

- At some point, we want our new nice_feature to be integrated into the master branch's code, too. Since their commits are separate, this is called *merging* the branches.

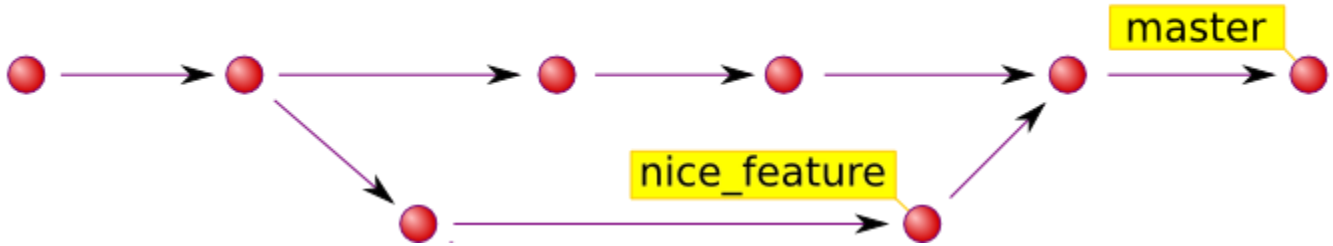


Figure from http://hades.github.io/2010/01/git-your-friend-not-foe-vol-2-branches/

A tour of branches in Git

- Notice that a *merge* involves <u>two</u> parent commits, instead of one! There is a possibility of a *conflict* if we edit the same files.
- Git is smart about merging, but sometimes you have to handle conflicts yourself.

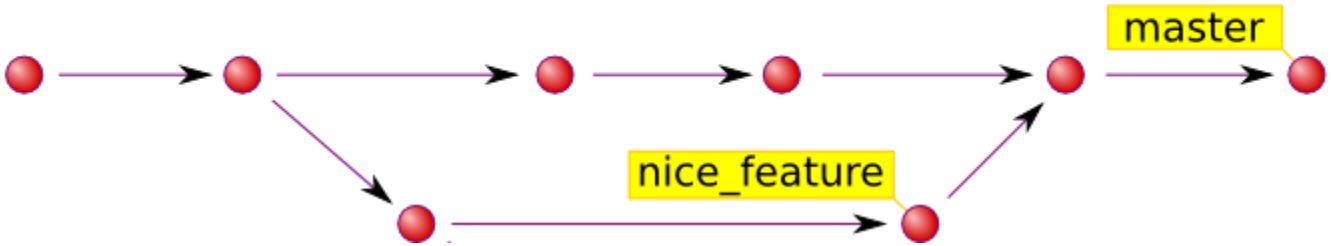


Figure from http://hades.github.io/2010/01/git-your-friend-not-foe-vol-2-branches/

A tour of branches in Git

- Notice that a *merge* involves <u>two</u> parent commits, instead of one! There is a possibility of a *conflict* if we edit the same files.
- Git is smart about merging, but sometimes you have to resolve conflicts yourself!

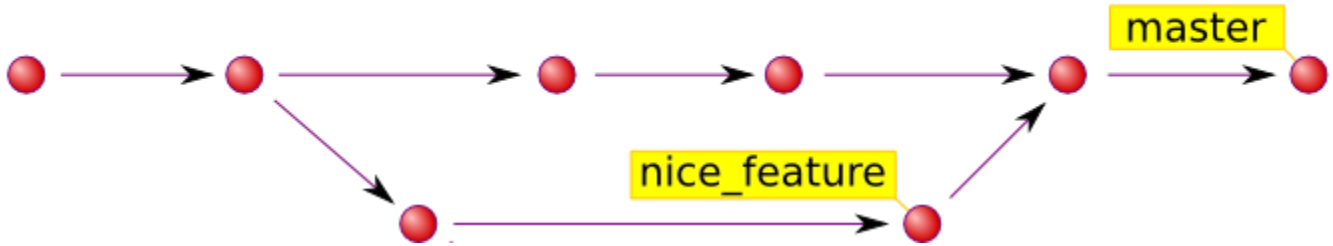


Figure from http://hades.github.io/2010/01/git-your-friend-not-foe-vol-2-branches/

A tour of branches in Git

- Of course, we're not limited to two branches…

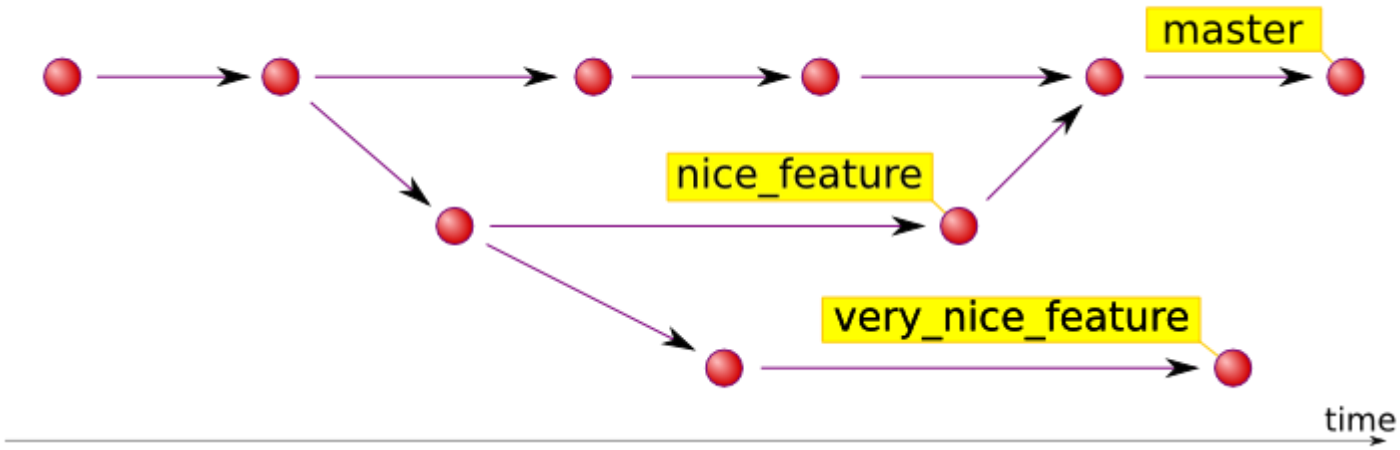


Figure from http://hades.github.io/2010/01/git-your-friend-not-foe-vol-2-branches/

A tour of branches in Git

- …or to merging only to one master branch

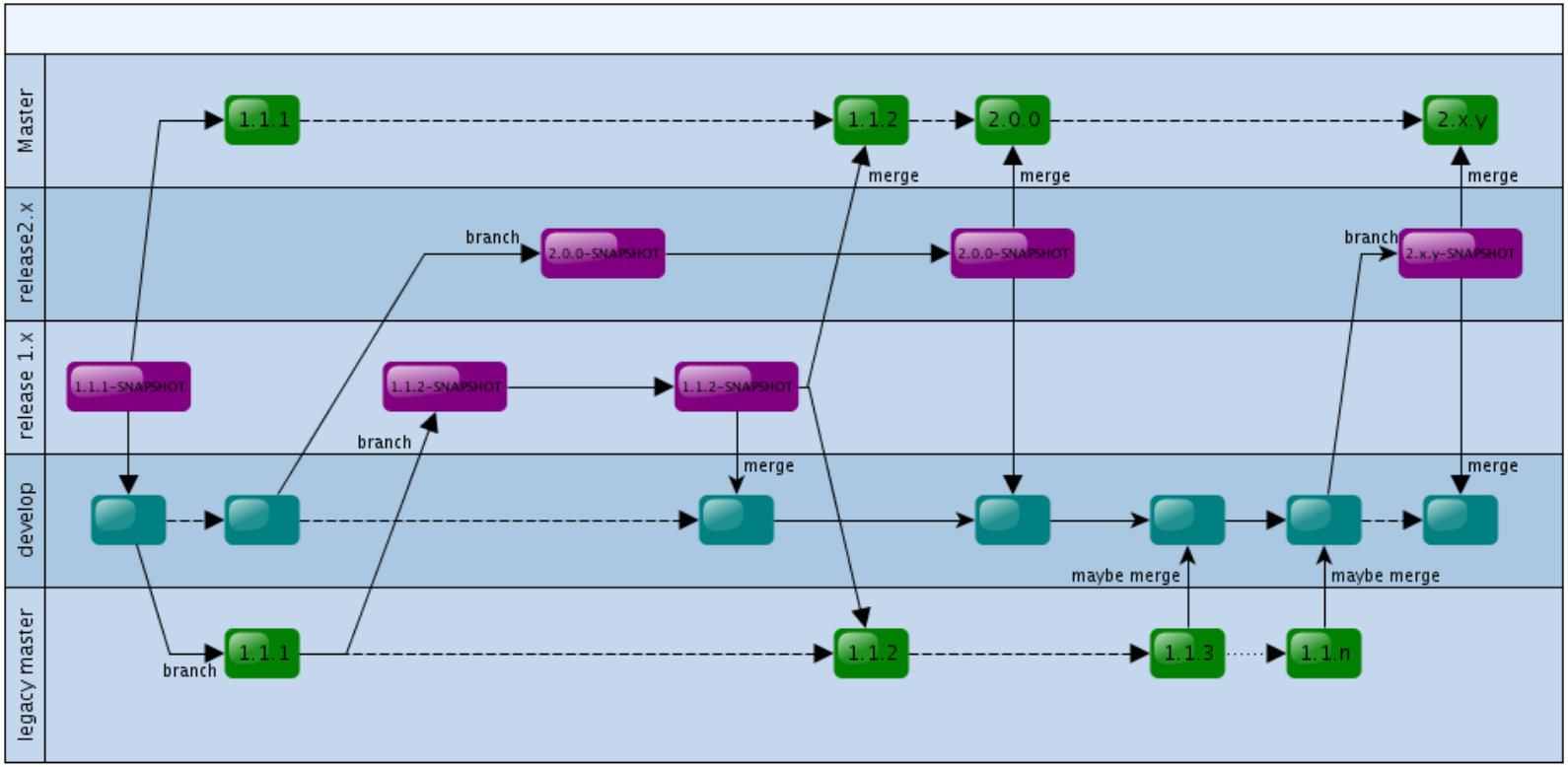


Image from OpenNCP project: https://openncp.atlassian.net/wiki/pages/viewpage.action?pageId=12812318

A tour of branches in Git

- Punchline: commits can have more than one parent, which lets us *branch* off and create "alternative histories."
- If we want to *merge* these histories, we can use the same property of commits to do so.
- These two facts together can create surprisingly powerful workflows
  - …but perhaps not ones you'll use very often

Things I *didn't* talk about

- Setting up connections to remote repos
- Reviewing history
  - We'll do a little of this in our hands-on
- 'Tagging' commits
- …
- Git can do a *lot*, but you might not need it
  - Try typing 'git help' or add '--help' to any cmd

Where to go to learn more

- The Git Book – https://git-scm.com/book/
- https://try.github.io
- Interactive demos on branching http://pcottle.github.io/learnGitBranching
- Tack '--help' onto any command!
- http://git-lectures.github.io/

Thank you!

Interactive demo

- If you haven't already, install git (or ssh to io)
- https://git-scm.com/download
- Make a directory called 'gitdemo'
- Open a terminal window and cd to the 'gitdemo' folder

- On Windows: use the "git bash here" option in the right-click menu to open a terminal window

git init

- Running git init will create a new repository in the .git/ folder.

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo
$ ls -la
total 6
drwxr-xr-x    2 James    Administ        0 Feb  4 18:06 .
dr-xr-xr-x   39 James    Administ    12288 Feb  4 18:07 ..

James@COLOSSUS /C/Users/James/Desktop/gitdemo
$ git init
Initialized empty Git repository in c:/Users/James/Desktop/gitdemo/.git/

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ ls -la
total 8
drwxr-xr-x    3 James    Administ        0 Feb  4 18:07 .
dr-xr-xr-x   39 James    Administ    12288 Feb  4 18:07 ..
drwxr-xr-x    9 James    Administ     4096 Feb  4 18:07 .git

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ _
```

git add + git commit

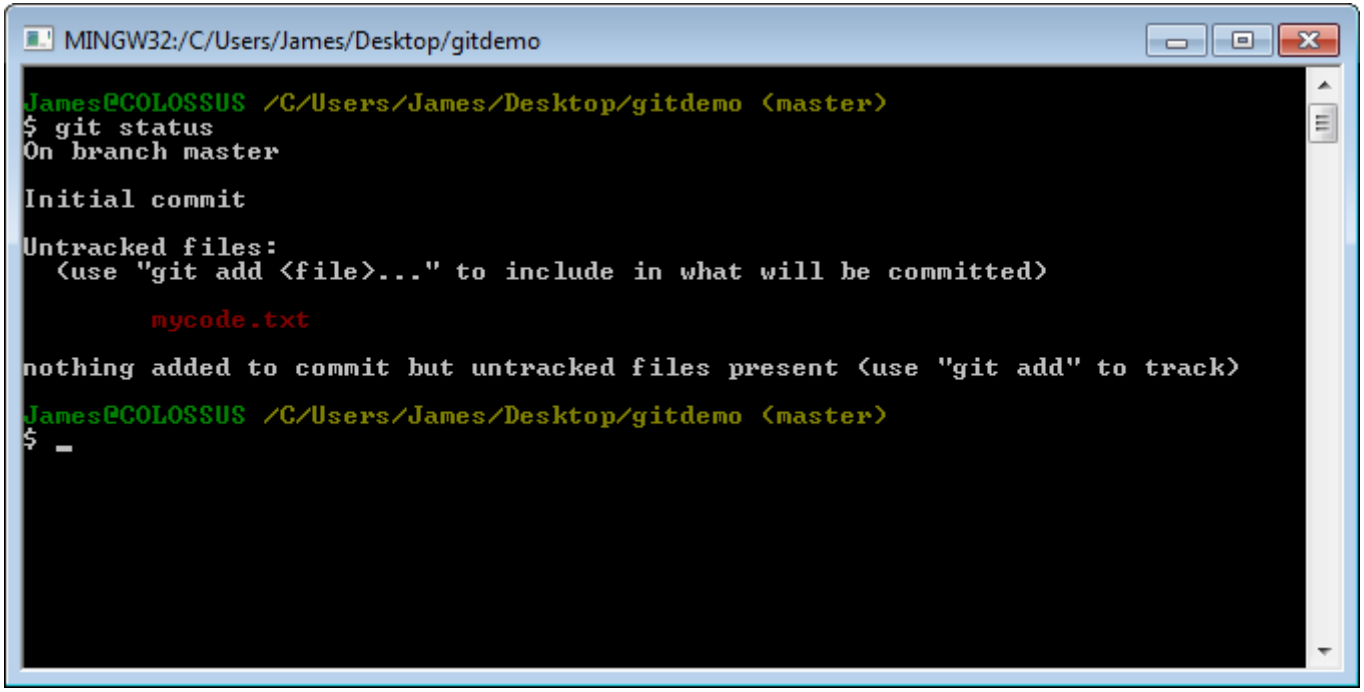
- Let's create the file mycode.txt to represent some code

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ echo "Hello, world" > mycode.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$
```

git add + git commit

- Running git status, notice that git sees this file, but tells us it won't be tracked



git add + git commit

- So let's tell it to track that file with git add and git commit the changes.

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git add mycode.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   mycode.txt


James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git commit -m "First version of our code"
[master (root-commit) 7f9d286] First version of our code
 1 file changed, 1 insertion(+)
 create mode 100644 mycode.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$
```

A second commit

- Let's edit the file and commit again
- Notice we have to add *again!*

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ echo "A second line of code" >> mycode.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git status
On branch master
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:   mycode.txt

no changes added to commit (use "git add" and/or "git commit -a")

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git add mycode.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git commit -m "We added a second line of code"
[master 1fd10ec] We added a second line of code
 1 file changed, 1 insertion(+)

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$
```

Adding files, deleting files

- Let's add a new file, edit the old one, and commit those changes.

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ echo "Here's a third line of code" >> mycode.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ echo "This is a brand new file" > newfile.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git add --all

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git commit -m "More edits to mycode.txt, and a new file"
[master d880db6] More edits to mycode.txt, and a new file
 2 files changed, 2 insertions(+)
 create mode 100644 newfile.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ _
```

Viewing history

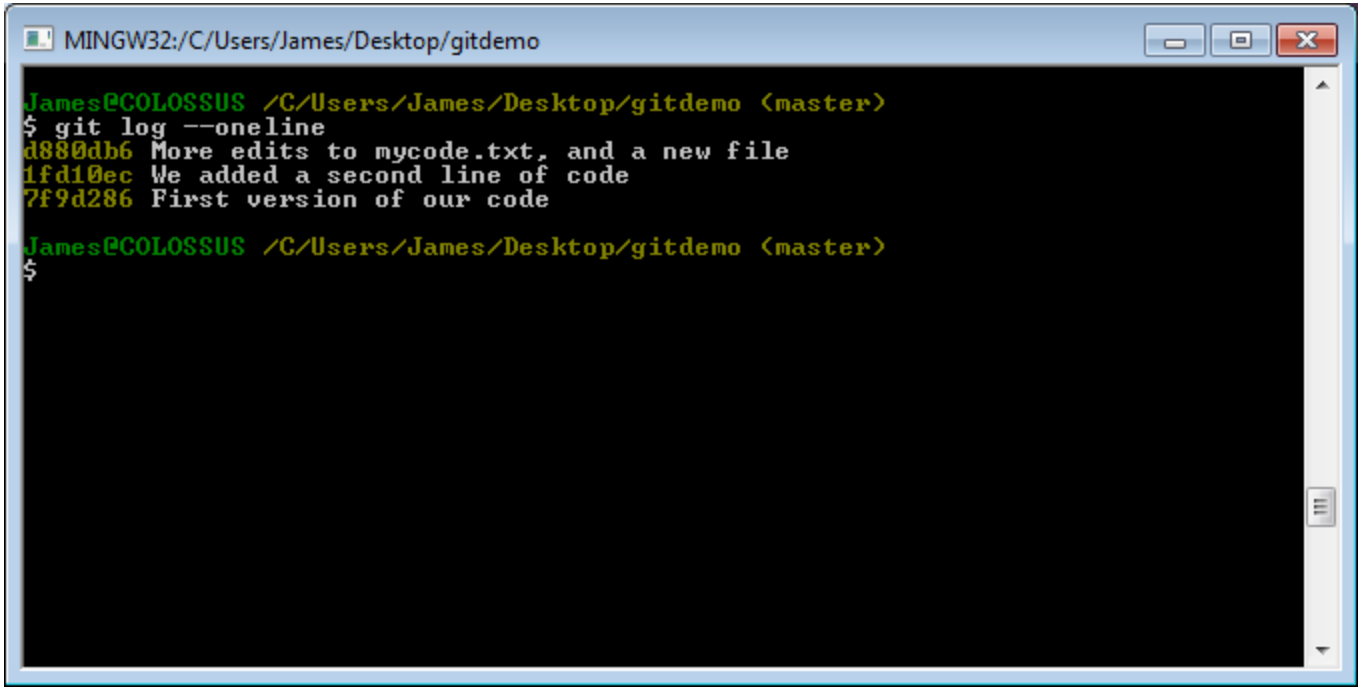
- Once we've committed, we can check our history with git log

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log
commit d880db61296b57706d4b73a247d432daca8aad66
Author: James <snoopjedi@gmail.com>
Date:   Fri Feb 5 00:20:48 2016 -0600

    More edits to mycode.txt, and a new file

commit 1fd10ec045c575c1b39dd6b26bd413a2b477727f
Author: James <snoopjedi@gmail.com>
Date:   Thu Feb 4 19:10:23 2016 -0600

    We added a second line of code

commit 7f9d2866f6353f5337c0620ebdfd36adb1e4ef36
Author: James <snoopjedi@gmail.com>
Date:   Thu Feb 4 19:09:32 2016 -0600

    First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$
```

Viewing history

- It's usually easier to view the log with the option '--oneline'



Time travel

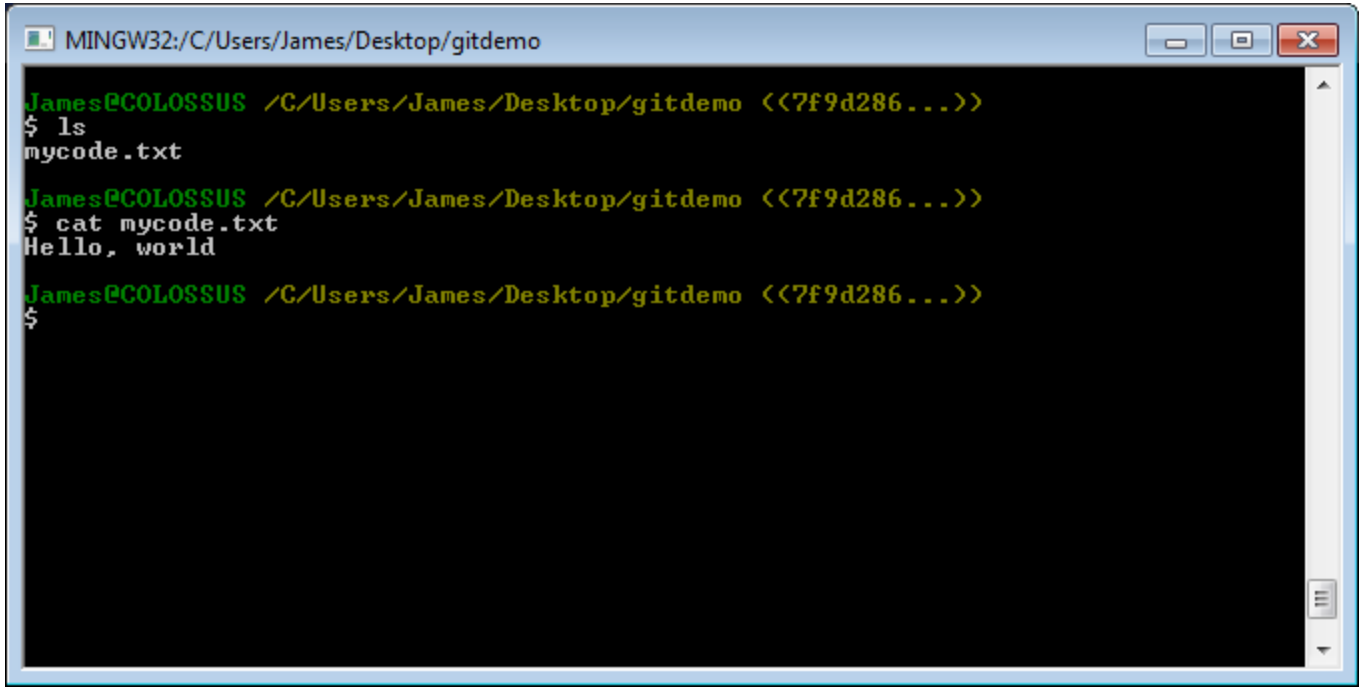
- If we want to look at an old commit, we can use git checkout

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
d880db6 More edits to mycode.txt, and a new file
1fd10ec We added a second line of code
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git checkout 7f9d286
Note: checking out '7f9d286'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 7f9d286... First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo ((7f9d286...))
$ _
```

Time travel

- If we want to look at an old commit, we can use git checkout



Time travel

- To leave the detached HEAD state, we use git checkout to return to ‘master’

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo ((7f9d286...))
$ git checkout master
Previous HEAD position was 7f9d286... First version of our code
Switched to branch 'master'

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ ls
mycode.txt  newfile.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ cat mycode.txt
Hello, world
A second line of code
Here's a third line of code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ _
```

‘Undoing’ commits

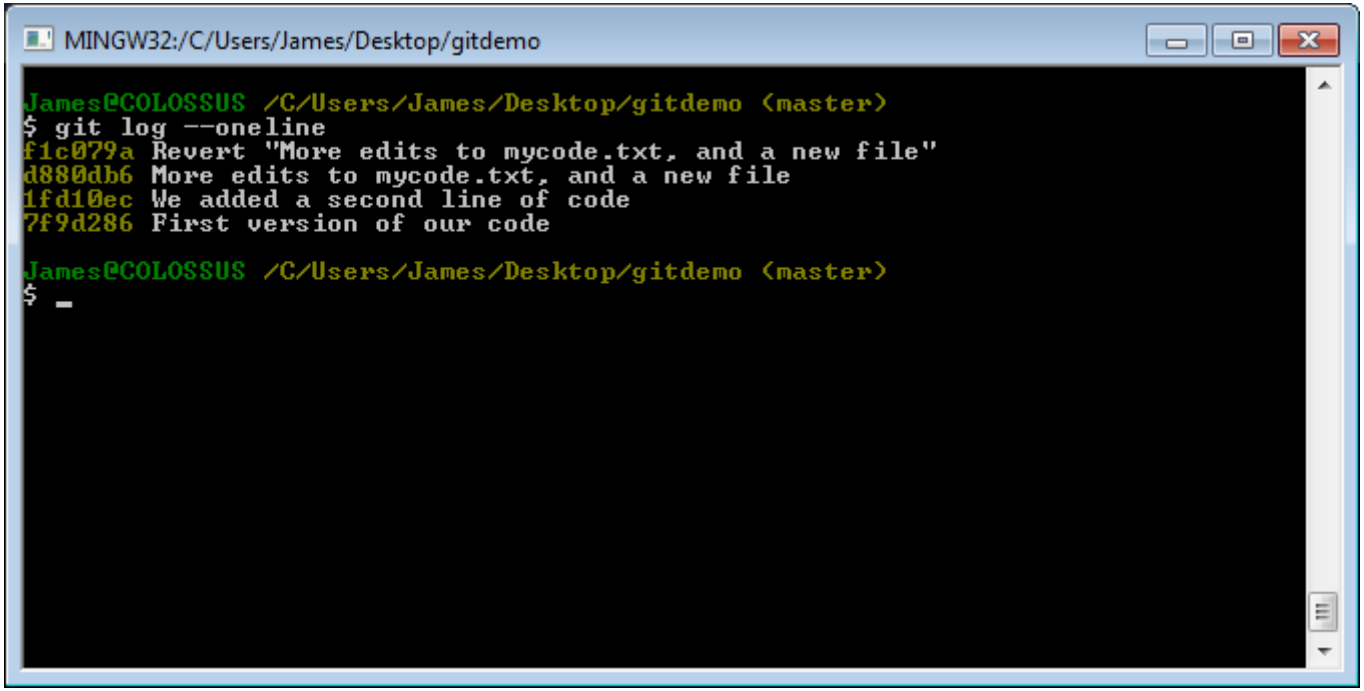
- Let’s use git revert to undo the changes one commit introduced.

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
d880db6 More edits to mycode.txt, and a new file
1fd10ec We added a second line of code
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git revert d880db6 --no-edit
[master f1c079a] Revert "More edits to mycode.txt, and a new file"
 2 files changed, 2 deletions(-)
 delete mode 100644 newfile.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ ls
mycode.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ cat mycode.txt
Hello, world
A second line of code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$
```

‘Undoing’ commits

- git revert undoes changes, but it preserves history by *making a new commit*



‘Undoing’ commits

- Let’s undo the revert without preserving history, using git reset

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
f1c079a Revert "More edits to mycode.txt, and a new file"
d880db6 More edits to mycode.txt, and a new file
1fd10ec We added a second line of code
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git reset --hard d880db6
HEAD is now at d880db6 More edits to mycode.txt, and a new file

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ ls
mycode.txt  newfile.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
d880db6 More edits to mycode.txt, and a new file
1fd10ec We added a second line of code
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ _
```

‘Undoing’ commits

- WARNING: since git reset *destroys* history, it’s usually bad for shared commits

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
f1c079a Revert "More edits to mycode.txt, and a new file"
d880db6 More edits to mycode.txt, and a new file
1fd10ec We added a second line of code
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git reset --hard d880db6
HEAD is now at d880db6 More edits to mycode.txt, and a new file

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ ls
mycode.txt  newfile.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
d880db6 More edits to mycode.txt, and a new file
1fd10ec We added a second line of code
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ _
```

A handy shorthand for git reset

- Using HEAD~N in git reset means “go to N commits behind where we are now”

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
d880db6 More edits to mycode.txt, and a new file
1fd10ec We added a second line of code
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git reset --hard HEAD~2
HEAD is now at 7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ cat mycode.txt
Hello, world

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ _
```

When git reset (or anything else) goes wrong

- If you think you've lost a commit, *don't panic*. Check git reflog

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git reflog
7f9d286 HEAD@{0}: reset: moving to HEAD~2
c0f41d1 HEAD@{1}: commit: More edits to mycode.txt, and a new file
57b81bf HEAD@{2}: commit: We added a second line of code
7f9d286 HEAD@{3}: reset: moving to HEAD~3
a1d6e2e HEAD@{4}: reset: moving to a1d6e2e
7f9d286 HEAD@{5}: reset: moving to HEAD~2
7c92a2b HEAD@{6}: reset: moving to HEAD~1
a1d6e2e

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$
```

When git reset (or anything else) goes wrong

- If the object is still in git reflog, we can get back to it with another git reset command.

MINGW32:/C/Users/James/Desktop/gitdemo

```
James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git log --oneline
7f9d286 First version of our code

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git reflog
7f9d286 HEAD@{0}: reset: moving to HEAD~2
c0f41d1 HEAD@{1}: commit: More edits to mycode.txt, and a new file
57b81bf HEAD@{2}: commit: We added a second line of code
7f9d286 HEAD@{3}: reset: moving to HEAD~3
a1d6e2e HEAD@{4}: reset: moving to a1d6e2e
7f9d286 HEAD@{5}: reset: moving to HEAD~2
7c92a2b HEAD@{6}: reset: moving to HEAD~1
a1d6e2e

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ git reset --hard c0f41d1
HEAD is now at c0f41d1 More edits to mycode.txt, and a new file

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$ ls
mycode.txt  newfile.txt

James@COLOSSUS /C/Users/James/Desktop/gitdemo (master)
$
```