In case of fire

1. git commit
2. git push
3. leave building
THE GIT SOURCE CODE MANAGER
A CASE STUDY

DR. NIELS JOUBERT
CS107
HOW CS107 PREPARED YOU TO UNDERSTAND REAL-WORLD CODE, BUILD BADASS SYSTEMS, INVENT THE FUTURE, TAKE OVER THE WORLD, AND COLONIZE THE GALAXY

DR. NIELS JOUBERT
CS107
INTRODUCING MYSELF

**Stanford Ph.D** in Computer Science, MSc in CS (Stanford), BSc in EECS (UC Berkeley).

I work as a **Technical Consultant**. I contribute to Open Source projects and advise several startups. aka, I’m a hired gun.


My research focus on **Enabling Cinematography** using **Micro Aerial Vehicles** (“Drones“)

**Full-Stack System Builder** From User Interfaces through Distributed Systems to Embedded Systems.

Worked on Web Technologies, Digital Textbooks, High Performance Computing, Massive-scale Software Provisioning, Satellite Systems
INTRODUCING MYSELF
WHERE DO YOU STORE YOUR CODE?
GIT: THE STUPID CONTENT TRACKER

https://trends.google.com/trends/explore?date=all&q=%2Fm%2F05vqwg,%2Fm%2F012ct9,%2Fm%2F08441_,%2Fm%2F08w6d6,%2Fm%2F09d6g&hl=en-US&tz=&tz=
MOST GIT TALKS WILL DO SOMETHING LIKE THIS...

GIT INIT
GIT ADD
GIT COMMIT
GIT PUSH GITGUB
Git Internals

----------------------------------

Git – the stupid content tracker

----------------------------------

"git" can mean anything, depending on your mood.

- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
- stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
- "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
- "goddamn idiotic truckload of sh*t": when it breaks

Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.
WALKTHROUGH OF GIT’S CORE PRINCIPLES
PRINCIPLE 1: UNIX DESIGN PHILOSOPHY

Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new “features”. —Doug McIlroy, 1978 Bell Labs
A FEW PRELIMINARIES
AKA: ASSUMPTIONS WE MAKE

Your project lives in a working directory - a root directory and tree of sub-directories containing code (text) files.

We (ahem, git) stores copies we call versions or snapshots of your code in a repository.
DESIGN PROMPT 1

Your customer wants a way to **save a snapshot of their code** whenever they feel like it. Let’s start by assuming their code is a single directory containing a few text files.

What are some good traits of a solution? Good features?
DESIGN PROMPT

Your customer wants a way to save a snapshot of their code whenever they feel like it. Let’s start by assuming their code is a single directory containing a few text files.

What are some good traits of a solution? Good features?

Never overwrite snapshots. Only write and read

We want to reduce duplicate data (within reason)

Track our progress.

Simple set of modular terminal commands (UNIX philosophy)
SIMPLE SOLUTION
COPY FILES TO HIDDEN DIRECTORY.

LET'S HACK THIS REAL QUICK
SIMPLE SOLUTION
COPY FILES TO HIDDEN DIRECTORY.

What’s good?

What sucks?
SIMPLE SOLUTION
COPY FILES TO HIDDEN DIRECTORY.

What’s **good**?

Dead simple.
No “delete” function. Only add.

What **sucks**?

Incredibly redundant
Incredibly crappy sense of tracking progress.
PRINCIPLE: IMMUTABLE STORE

Can add data. Can extract data. Can’t modify (mutate) data.
WHAT CAN WE DO ABOUT REDUNDANCY?

Any ideas?
REMEMBER HASHES?

A function that:

- Summarizes arbitrarily long input to a fixed length output
- Guaranteed to have same output for same input
- Highly unlikely for different inputs to have same output

```
$ shasum ontheroad.txt
d6561b576870e883d1cc57711966bc4ca9d162cb  ontheroad.txt
$_
```
MORE COMPLEX SOLUTION
STORE CONTENT SEPARATE FROM FILE

Store the content of every file in a special hidden directory.

Name these files using the hash of the content.

Store the filenames separately.

Each file just contains the hash of the content.
MORE COMPLEX SOLUTION
STORE CONTENT SEPARATE FROM FILE

Store the content of every file in a special hidden directory.

Name these files using the hash of the content.

Store the filenames separately.

Each file just contains the hash of the content.

PRINCIPLE: USING HASHES AS POINTERS
“CONTENT-ADDRESSABLE FILE SYSTEM”
LET’S HACK THIS TOGETHER REAL QUICK
YOU KNOW THE CONCEPTS INSIDE GIT!

Git is just:

An immutable, content-addressable file system

and a tree structure storing history
LET’S GET INTO HOW GIT WORKS
Content-addressable file system stored in .git/objects

Key: SHA-1 Hash of the file contents

Value: Contents of the file (zlib-compressed)
GIT OBJECTS: BLOB

Wanna hang on to the content of an individual file in a blob object:

```bash
git hash-object -w ontheroad.txt
```

Wanna see what’s in the object store?

```bash
find -f .git/objects
```

Wanna pull it out?

```bash
git cat-file -p hash
```
EXAMPLE: EVERYTHING IS NAMED BY ITS HASH
**GIT OBJECTS: TREE**

Tree objects stores your filenames and directory structure.

Create tree objects using the stage ("index")

```
git update-index --add --cacheinfo mode hash name
git write-tree
```

```
git cat-file the result: it's just a text file!
```
HOW DO I MAKE A SNAPSHOT?

Are we ready to recreate my snapshot.sh script?
HOW DO I MAKE A SNAPSHOT?

For each file:

```
git hash-object -w
```

```
Git update-index hash, filename
```

```
Git write-tree
```
We have to remember the hash of the root tree for every version.

We aren’t tracking the sequence of trees: which tree came first?
GIT OBJECTS: A FIRST COMMIT

Commit object stores:

- Hash of previous root tree
- Hash of current root tree
- A bit of metadata

`git commit-tree`
QUESTION: WHAT HAPPENS IF I CHANGE ONE LINE IN A LARGE TEXT FILE, AND RE-COMMIT?
In our second commit, we save a *parent pointer*

git commit-tree *tree-hash* -p *parent-commit-hash*
STRUCTURE SO FAR

2342075f
First commit

cfef27a0
Second commit

4b235421
Tree

4cb92c71
ontheroad.txt

f6f58185
JohnBarleycorn.txt

1206d2d5
ontheroad.txt
WHAT WE HAVE SO FAR

An immutable content-addressable file system storing:

Raw content

Graph: directory trees and snapshot history
WHAT’S THE MOST PAINFUL ASPECT SO FAR?
HASHES
Wouldn’t it be nice to have human-readable names for important commits.

Yup. That’s a reference or a branch, and they live in .git/refs/heads/master.
Git convention:

The main reference (branch) is called "master"

The reference corresponding to your current working directory is called "head"

Whenever we take a snapshot, we update master to point to the latest snapshot.
GIT REFERENCES ARE MUTABLE! OUR FIRST AND ONLY MUTABLE STRUCTURE
VISUALLY. (SEE GITK)

```
2342075f
First commit

4b235421
Tree

47130de9
Tree

4cb92c71
ontheroad.txt

f6f58185
JohnBarleycorn.txt

1206d2d5
ontheroad.txt
```
FINALLY WE CAN TAKE A SNAPSHOT
PLUMBING COMMANDS:

Take your files

Hash files into blob objects (only changed files are added)

Create tree objects capturing filenames

Create a commit object (stores the root of this snapshot)

Update your current reference/branch to this commit
FINALLY WE CAN TAKE A SNAPSHOT
PORCELAIN COMMANDS:

```plaintext
git add

git commit
```
REFERENCES ("BRANCHES") ENABLE ALL SORTS OF COOL STUFF

With one developer:

I can keep my master reference pointing to a snapshot of working code. I can have a development reference pointing to my latest snapshot of experimental code. I can easily jump between them.

Collaboration:

I can store different references for different people. eg: niels/master and julie/master
COLLABORATING WITH GIT

Danger: gonna get real hand-wavey here
PRINCIPLES: GIT IS DECENTRALIZED

Every developer has a copy of the entire repository.

To share code, simply combine the two data-stores, and keep their references separate from yours.

It’s all immutable and content-addressable! Just copy all their objects into yours!
REFERENCES: EXAMPLE
WHAT IS THE ECOSYSTEM ENABLED BY
GIT’S DESIGN?
LIGHTWEIGHT VERSIONING CODE IS A NO-BRAINER

VERSION

ALL THE THINGS
Lightweight branching: no big copy necessary, just a new reference.
Decentralized: Everyone has their own full repository.

Massive impact in the Open Source world! Erases barriers to contribution. No need to manage access control between users. Everyone gets to play.
DECENTRALIZED EMPOWERS COLLABORATION

Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside millions of other developers.
DECENTRALIZED EMPOWERS COLLABORATION

https://github.com/njoubert/node-groupme/network
GIT’S CONTROVERSIAL HISTORY

THE PEOPLE BEHIND THE CODE
WHY IS IT CALLED GIT?

Dictionary

**git**

/git/

noun  BRITISH  informal

an unpleasant or contemptible person.

Translations, word origin, and more definitions
WHY IS IT CALLED GIT?
LINUS FEUDS WITH TRIDGE?

Larry McVoy convinces Linux to use BitKeeper.

BitKeeper is free (beer) but not free (speech), lacks features the kernel developers want.

Tridge doesn’t wait for negotiations, reverse-engineers BitKeeper protocol.

Linus gets incredibly annoyed: "You did not create something better (or equivalent) that we could use. You basically ruined our relationship with BitMover for no good reason”

Linux builds git in 10 days, names it after Tridge

https://en.wikipedia.org/wiki/Andrew_Tridgell
WHY IS IT CALLED GIT?
LINUS FEUDS WITH TRIDGE?

Business ➔ Policy

Torvalds knifes Tridgell
Kernel source row turns nasty
By Andrew Orlowski | 14 Apr 2005 at 20:32

Linux founder Linus Torvalds has followed up his weekend condemnation of reverse engineering with an astonishing personal attack on the integrity of one of the most respected figures in the open source community, menko author and Samba co-lead Andrew Tridgell.

Torvalds accuses Tridgell of playing dirty tricks with his proprietary source code tool of choice, Bitkeeper and destabilizing the product. These are serious accusations to make.

Tridgell, we've learned, was attempting to gain knowledge of the Bitkeeper protocol on the wire, so he could allow the Linux kernel developers to retrieve their source code metadata from the dark dungeons of Larry McVoy's back garden (i.e. Bitkeeper). This metadata is one piece of information that Bitkeeper regards as proprietary to itself – so, if you're not a paid-up Bitkeeper licensee, you never get to see it. But kernel developers like to have this information, and Tridgell was trying to open up the possibility for a third-party client to work with Bitkeeper.

Torvalds strongly disputes this characterization of Bitkeeper, but McVoy has made his position clear: no Bitkeeper license, no metadata.

"You need to understand that this is all you get, we're not going to extend [the Bitkeeper] license to the kernel," McVoy said.

Torvalds also accused Tridgell of trying to use the pay for Bitkeeper to manage the Linux kernel source code (pitching other kernel developers to follow suit), but last week its owner, Bitkeeper CEO Larry McVoy, yanked the license, pushing Torvalds to look for an alternative. He's now going to write his own. For this inconvenience, he blames Tridgell.

It's a conflict of interest, treated with disdain by the open source community, and he's decided to break the rules.
LINUS TORVALDS ON BUILDING GIT

You can actually see how it all took shape in the git source code repository, except for the very first day or so. It took about a day to get to be “self-hosting” so that I could start committing things into git using git itself, so the first day or so is hidden, but everything else is there. The work was clearly mostly during the day, but there’s a few midnight entries and a couple of 2 a.m. ones. The most interesting part is how quickly it took shape; the very first commit in the git tree is not a lot of code, but it already did the basics – enough to commit itself. The trick wasn’t really so much the coding but coming up with how it organizes the data.

So I’d like to stress that while it really came together in just about ten days or so (at which point I did my first *kernel* commit using git), it wasn’t like it was some kind of mad dash of coding. The actual amount of that early code is actually fairly small, it all depended on getting the basic ideas right. And that I had been mulling over for a while before the whole project started. I’d seen the problems others had. I’d seen what I wanted to avoid doing.
WHY IS GIT HARD?
WHY IS GIT HARD?

http://justinhileman.info/article/git-pretty/
WHY IS GIT HARD?

Git is, really, an API to build version control workflows. Doesn’t care how you use it, supports many workflows, can even be used for all kinds of non-source-code-management stuff

eg: digital billboard data management at reactrix.com
WHY IS GIT HARD?

"You can do a lot of things with git, and many of the rules of what you *should* do are not so much technical limitations but are about **what works well when working together with other people**. So git is a very powerful set of tools, and that can not only be overwhelming at first, it also means that you can often do the same (or similar) things different ways, and they all “work.” Generally, the best way to learn git is probably to first only do very basic things and not even look at some of the things you can do until you are familiar and confident about the basics”

IMHO, this is a good reason to know **how git works**.
Git stores snapshots of your code in hidden directory called a repository. This repository is a content-addressable, immutable datastore containing blobs, trees, and commits.

Content is addressed by its hash
Trees and commits uses hashes as pointers

Every snapshot (except the first) has a parent snapshot. You can walk these parent pointers to explore the history of your project. History can split (two different commits pointing to the same parent).

Human-readable references point to commits. These references track a branch of development over time.
I’m sorry I did not teach you how to use Git!

The plumbing commands I showed you is impractical for day-to-day use.

And there’s a lot more to learn about how to use Git.

But if you have one take-away: large successful systems use the same core concepts you’ve learned in CS107.
FIND ME ON GITHUB

https://github.com/njoubert/

Playing With Fire

Programatically-controlled Propane-based Flame Effects

The goal of this project is to increase the number of flame effects in the festival circuit. We'd like to make it easier for others to create their own propane-based flame effects.
RESOURCES


Gitlet.js http://gitlet.maryrosecook.com/docs/gitlet.html


Git From the Bits Up https://www.youtube.com/watch?v=MYP56QJpDr4

What are your questions?