Learning Git± in Reverse

A Backwards Introduction to
the “information manager from hell” [e83c51633]

Kenny Ballou

California State University San Marcos

Spring 2025

About You

About You

- Heard of Git$\pm$ before

About You

- Heard of Git± before
- Used Git± before

About You

- Heard of Git± before
- Used Git± before
- Use Git± daily

About You

- Heard of Git± before
- Used Git± before
- Use Git± daily
- Consider yourself a Git± expert

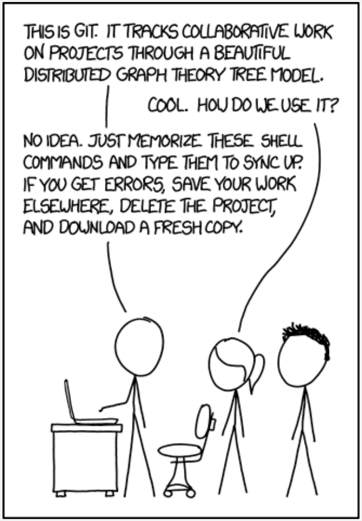

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Figure: XKCD on Git[8]

Git±

What is Git±?

Git±

What is Git±?

- Distributed Version Control System (D-VCS)

Git±

What is Git±?

- Distributed Version Control System (D-VCS)
- “A way to manage code”

Git±

What is Git±?

- Distributed Version Control System (D-VCS)
- “A way to manage code”
- “My preferred VCS tool”

Git±

What is Git±?

- Distributed Version Control System (D-VCS)
- “A way to manage code”
- “My preferred VCS tool”
- The “information manager from hell”

Git±

What is Git±?

- Distributed Version Control System (D-VCS)
- "A way to manage code"
- "My preferred VCS tool"
- The "information manager from hell"
- A distributed DAG
  - "A Graph Tree"

Git±

What is Git±?

- Distributed Version Control System (D-VCS)
- “A way to manage code”
- “My preferred VCS tool”
- The “information manager from hell”
- A distributed DAG
    - “A Graph Tree”
- An object store

Git±

What is Git±?

- Distributed Version Control System (D-VCS)
- "A way to manage code"
- "My preferred VCS tool"
- The "information manager from hell"
- A distributed DAG
  - "A Graph Tree"
- An object store
- A content addressable filesystem

Git±

What is Git±?

- Distributed Version Control System (D-VCS)
- “A way to manage code”
- “My preferred VCS tool”
- The “information manager from hell”
- A distributed DAG
    - “A Graph Tree”
- An object store
- A content addressable filesystem
- A key-value store

Git±

What does Git± store?

Git±

What does Git± store?

- Objects

Git±

What does Git± store?

- Objects
- Commits

Git±

What does Git± store?

- Objects
- Commits
- Code

Git±

What does Git± store?

- Objects
- Commits
- Code
- “Packs”

Git Definitions

- Objects
- Trees
- Commits
- References

Git Definitions

“It’s turtles all the down”

- Objects
- Trees $\leftarrow$ objects
- Commits $\leftarrow$ objects

Initializing a Repository the Hard Way

```
% cd /tmp
% mkdir -p foo/.git/objects/{info,pack}
% mkdir -p foo/.git/hooks
% mkdir -p foo/.git/refs/{tags,heads}
% echo "ref: refs/heads/trunk" > foo/.git/HEAD
% cat << EOF > foo/.git/config
>[core]
>        repositoryformatversion = 0
>        filemode = true
>        bare = false
>        logallrefupdates = true
EOF
% cd foo
```

Initialization Results

```
± find .git
.git
.git/objects
.git/objects/info
.git/objects/pack
.git/config
.git/HEAD
.git/hooks
.git/refs
.git/refs/tags
.git/refs/heads
```

Git Objects

- ZLIB compressed blob
- Dumb containers, storing provided content
- Created using the `git-hash-object` plumbing command

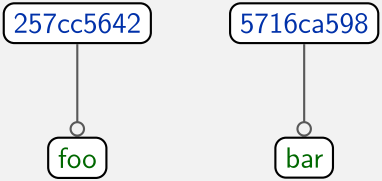
git-hash-object

```
± echo 'foo' | git hash-object --stdin
257cc5642cb1a054f08cc83f2d943e56fd3ebe99
± echo 'bar' | git hash-object --stdin
5716ca5987cbf97d6bb54920bea6adde242d87e6
```

git-hash-object

```
± echo 'foo' | git hash-object -w --stdin
257cc5642cb1a054f08cc83f2d943e56fd3ebe99
± echo 'bar' | git hash-object -w --stdin
5716ca5987cbf97d6bb54920bea6adde242d87e6
± find .git/objects -type f
.git/objects/57/16ca5987cbf97d6bb54920bea6adde242d87e6
.git/objects/25/7cc5642cb1a054f08cc83f2d943e56fd3ebe99
```

Git Objects



git-cat-file

```
± git cat-file -p 257cc5642cb1a054f08cc83f2d943e56fd3ebe99
foo
± git cat-file -p 5716ca5987cbf97d6bb54920bea6adde242d87e6
bar
```

Raw Access to Git Objects

```
± cat .git/objects/25/7cc5642cb1a054f08cc83f2d943e56fd3ebe99 \
  | zlib-flate -uncompress
blob 4foo
```

Git Object Limitations

- Remembering 40 character SHA's is hard
- What about file names?

Git Trees

- ZLIB compressed blobs
- Contain references to files and other trees
- Created using the `git-update-index` and `git-write-tree`

git-update-index and git-write-tree

```
± git update-index --add --cacheinfo 100644 \
    257cc5642cb1a054f08cc83f2d943e56fd3ebe99 foo.txt
± git write-tree
fcf0be4d7e45f0ef9592682ad68e42270b0366b4
± git cat-file -p fcf0be4d7e45f0ef9592682ad68e42270b0366b4
100644 blob 257cc5642cb1a054f08cc83f2d943e56fd3ebe99        foo.txt
± git cat-file -t fcf0be4d7e45f0ef9592682ad68e42270b0366b4
tree
```

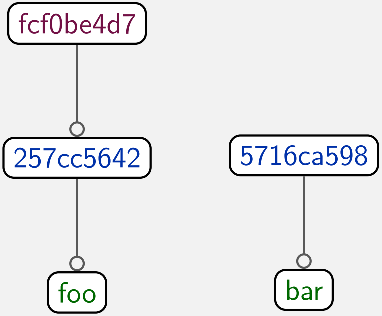
Git± Tree File Modes

100644	Regular file, *nix permissions 0644
100755	Regular file, *nix permissions 0755, e.g., executable
120000	Symbolic link
160000	“Gitlink”, object from another repository, `fast-import`
040000	Subdirectory, `fast-import`

Current Git Objects

```
± find .git/objects -type f
.git/objects/fc/f0be4d7e45f0ef9592682ad68e42270b0366b4
.git/objects/57/16ca5987cbf97d6bb54920bea6adde242d87e6
.git/objects/25/7cc5642cb1a054f08cc83f2d943e56fd3ebe99
```

Current Git Objects



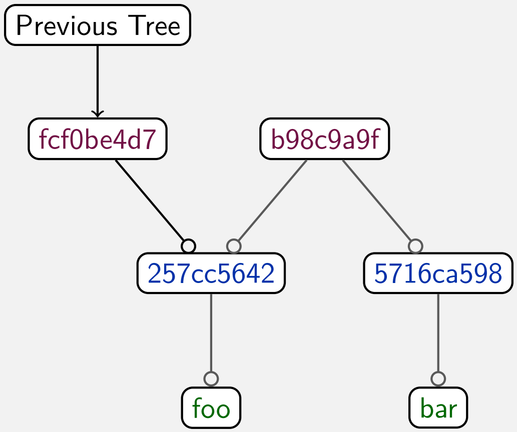
Adding bar.txt

```
± git update-index
± git update-index --add --cacheinfo 100644 \
    5716ca5987cbf97d6bb54920bea6adde242d87e6 bar.txt
± git write-tree
b98c9a9f9501ddcfcbe02a9de52964ed7dd76d5a
± git cat-file -p b98c9a9f9501ddcfcbe02a9de52964ed7dd76d5a
100644 blob 5716ca5987cbf97d6bb54920bea6adde242d87e6        bar.txt
100644 blob 257cc5642cb1a054f08cc83f2d943e56fd3ebe99        foo.txt
```

Git Objects

```
± find .git/objects -type f
.git/objects/b9/8c9a9f9501ddcfcbe02a9de52964ed7dd76d5a
.git/objects/fc/f0be4d7e45f0ef9592682ad68e42270b0366b4
.git/objects/57/16ca5987cbf97d6bb54920bea6adde242d87e6
.git/objects/25/7cc5642cb1a054f08cc83f2d943e56fd3ebe99
```

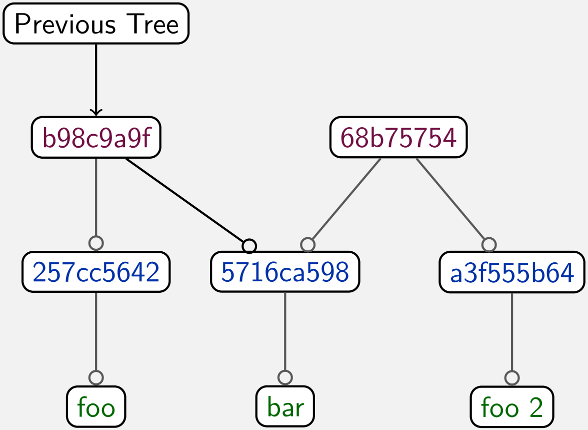
Git Objects, Updated



Modifying Files

```
± echo 'foo 2' > foo.txt
± git hash-object -w foo.txt
a3f555b643cbba18c0e69c82d8820c7487cebe15
± git update-index
± git update-index --add foo.txt
± git write-tree
68b757546e08c1d9033c8802e4de1c0d591d90c8
± git cat-file -p 68b757546e08c1d9033c8802e4de1c0d591d90c8
100644 blob 5716ca5987cbf97d6bb54920bea6adde242d87e6        bar.txt
100644 blob a3f555b643cbba18c0e69c82d8820c7487cebe15        foo.txt
```

Git Objects, Updated



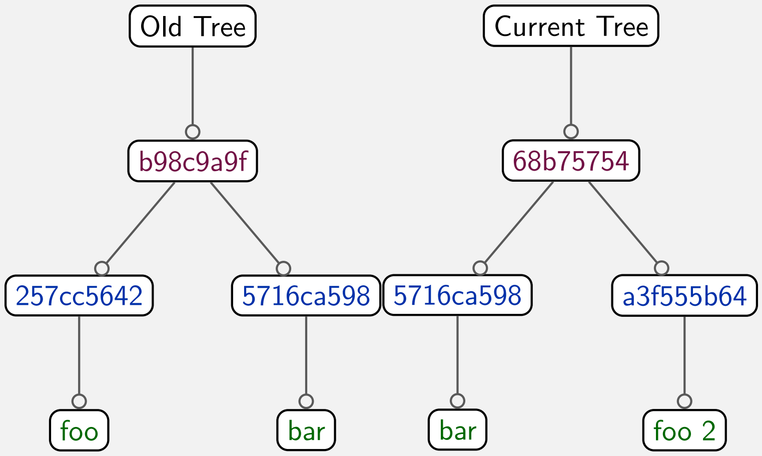
Modifying Files

Current Objects

```
± find .git/objects -type f
.git/objects/68/b757546e08c1d9033c8802e4de1c0d591d90c8
.git/objects/a3/f555b643cbba18c0e69c82d8820c7487cebe15
.git/objects/b9/8c9a9f9501ddcfcbe02a9de52964ed7dd76d5a
.git/objects/fc/f0be4d7e45f0ef9592682ad68e42270b0366b4
.git/objects/57/16ca5987cbf97d6bb54920bea6adde242d87e6
.git/objects/25/7cc5642cb1a054f08cc83f2d943e56fd3ebe99
```

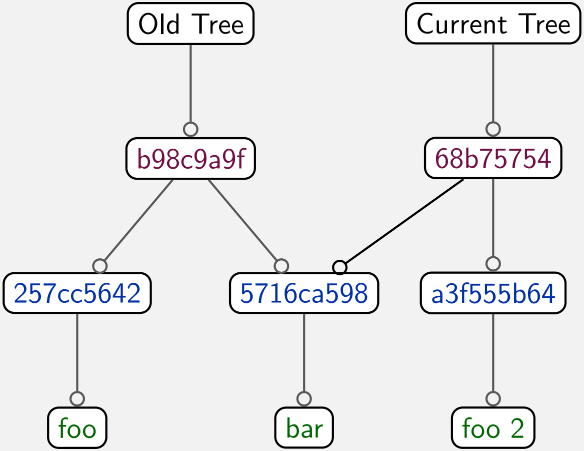
Modifying Files

Current Objects



Current Objects

The Beginnings of a DAG



Limitation of Git± Trees

- Remembering SHA's is *still* hard
- No metadata about who, when, and why

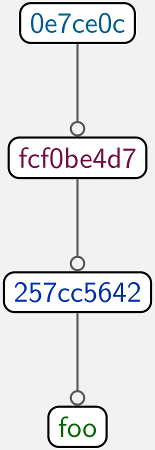
Git± Commit Objects

- ZLIB compressed blob
- Stores metadata about changes
- Stores a reference to the tree being saved
- Created using `git-commit-tree`

git-commit-tree

```
± echo 'our first commit' | git commit-tree \
    fcf0be4d7e45f0ef9592682ad68e42270b0366b4
0e7ce0ccc4dc5509e6730acf44c87156d7f066be
± git cat-file -p 0e7ce0ccc4dc5509e6730acf44c87156d7f066be
tree fcf0be4d7e45f0ef9592682ad68e42270b0366b4
author kballou <kballou@devnulllabs.io> 1489182707 -0700
committer kballou <kballou@devnulllabs.io> 1489182707 -0700

our first commit
```

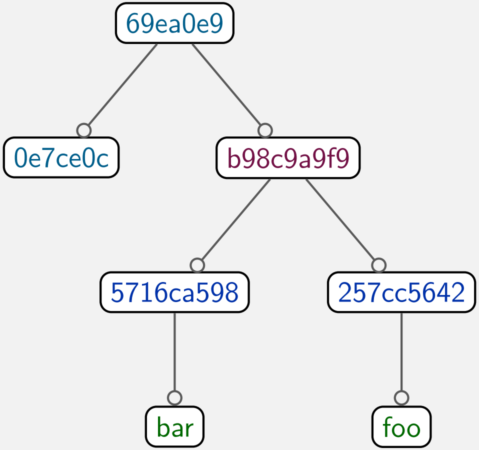
git-commit-tree



git-commit-tree

```
± echo 'our second commit' | git commit-tree \
    -p 0e7ce0ccc4dc5509e6730acf44c87156d7f066be \
    b98c9a9f9501ddcfcbe02a9de52964ed7dd76d5a
69ea0e93708eb39eacdf5dd8be9d1fc0a371fe1e
± git cat-file -p 69ea0e93708eb39eacdf5dd8be9d1fc0a371fe1e
tree b98c9a9f9501ddcfcbe02a9de52964ed7dd76d5a
parent 0e7ce0ccc4dc5509e6730acf44c87156d7f066be
author kballou <kballou@devnulllabs.io> 1489182899 -0700
committer kballou <kballou@devnulllabs.io> 1489182899 -0700

our second commit
```

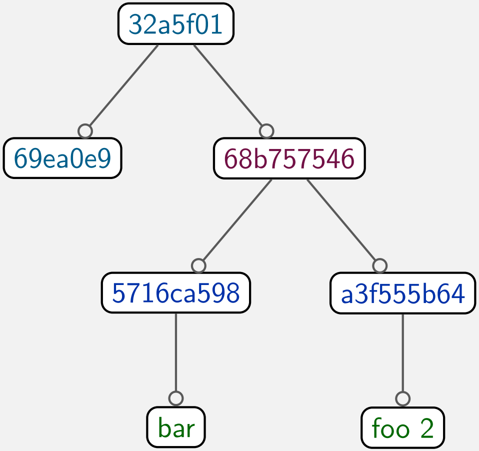
git-commit-tree



git-commit-tree

```
± echo 'our third commit' | git commit-tree \
    -p 69ea0e93708eb39eacdf5dd8be9d1fc0a371fe1e
    68b757546e08c1d9033c8802e4de1c0d591d90c8
32a5f01ac61c86d70c5b38bc5f43eb7cc4f27521
± git cat-file -p 32a5f01ac61c86d70c5b38bc5f43eb7cc4f27521
tree 68b757546e08c1d9033c8802e4de1c0d591d90c8
parent 2de9adf2b64be21358265a9fd61f70b87a200c20
author kballou <kballou@devnulllabs.io> 1489183140 -0700
committer kballou <kballou@devnulllabs.io> 11489183140 -0700

our third commit
```

git-commit-tree

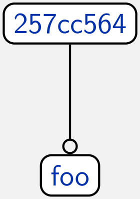


Git± History

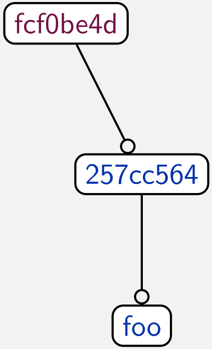
```
± git log --stat --oneline 32a5f01a

32a5f01a our third commit
 foo.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
69ea0e9 our second commit
 bar.txt | 1 +
 1 file changed, 1 insertion(+)
0e7ce0c our first commit
 foo.txt | 1 +
 1 file changed, 1 insertion(+)
```

Git± Thus Far

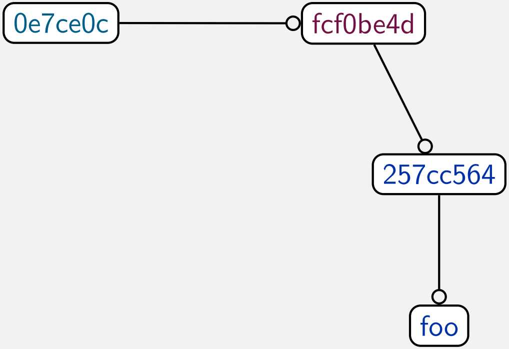
Git± Thus Far



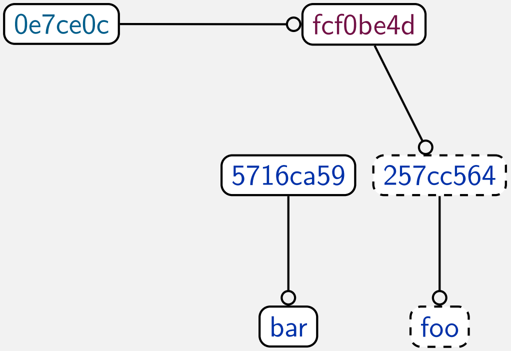
Git± Thus Far



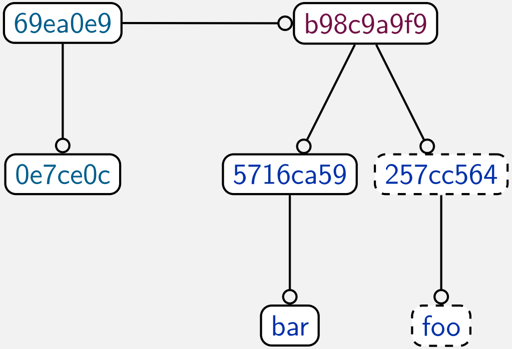
Git± Thus Far



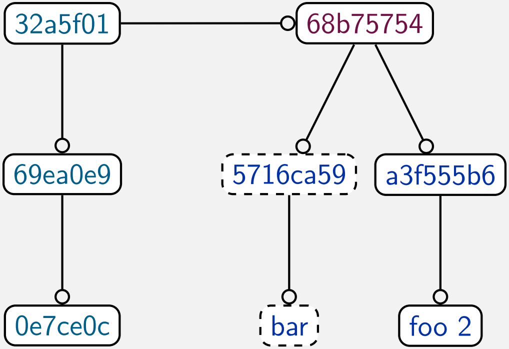
Git± Thus Far



Git± Thus Far



Git± Thus Far


32a5f01
68b75754
69ea0e9
5716ca59
a3f555b6
0e7ce0c
bar
foo 2

Git± Packfiles

Git± Packfiles

- Tight Object Storage

Git± Packfiles

- Tight Object Storage
- “Packs” Objects Together

Git± Packfiles

- Tight Object Storage
- “Packs” Objects Together
- Adds quick, indexed access to objects

Git± Packfiles

- Tight Object Storage
- “Packs” Objects Together
- Adds quick, indexed access to objects
- Motivated for network and access efficiency
  - Not Disk Space Efficiency

Git± Packfiles

- Tight Object Storage
- “Packs” Objects Together
- Adds quick, indexed access to objects
- Motivated for network and access efficiency
    - Not Disk Space Efficiency
- Created automatically

Creating Packfiles

- `git-pack-objects`

Creating Packfiles

- `git-pack-objects`
- `git-gc`

Creating Packfiles

git-pack-objects

```
± git rev-list --objects --all | head -1 | \
  git pack-objects --stdout | xxd
Counting objects: 1, done.
Total 1 (delta 0), reused 0 (delta 0)
00000000: 5041 434b 0000 0002 0000 0001 950e 789c  PACK..........x.
00000010: 9d8c 510e c220 1005 ff39 c55e 40b3 0894  ..Q.. ...9.^@...
00000020: 2531 c6ab 00bb 4d1b 6931 089e 5f8d 9ec0  %1....M.i1.._...
00000030: bf37 93cc eb4d 0426 4ade 7967 2741 ca9a  .7...M.&J.yg'A..
00000040: 031a 9389 f024 9645 6764 173e 3293 bac7  .....$.Egd.>2...
00000050: 267b 8729 4844 09c6 2349 326f c83c 3b66  &{.)HD..#I2o.<;f
00000060: 4a12 58cf 19a3 f17a 162d 2a8e bed4 06b7  J.X....z.-*.....
00000070: 144b a903 cebf 7165 79ee a394 12d3 e3b8  .K....qey.......
00000080: d60b 684b 4193 d116 e180 1e51 e5ba 6d6b  ..hKA......Q..mk
00000090: eff2 4fab ea68 d097 b531 7c6f d40b 6bc6  ..O..h...1|o..k.
000000a0: 4735 ffc0 30ce a4c0 140a a02e 648f 775c  G5..0.......d.w\
000000b0: 1fd1 654d a59e                           ..eM..
```

Creating Packfiles

git-pack-objects

```
± git rev-list --objects --all | head -1 | \
  git pack-objects test
Counting objects: 1, done.
Total 1 (delta 0), reused 0 (delta 0)
± ls test-*
test-ffc030cea4c0140aa02e648f775c1fd1654da59e.idx
test-ffc030cea4c0140aa02e648f775c1fd1654da59e.pack
```

Creating Packfiles
git-gc

```
± git gc
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (9/9), done.
Total 9 (delta 0), reused 0 (delta 0)
± find .git/objects -type f
.git/objects/info/packs
.git/objects/pack/pack-6b3ea4129144c819898dd16a08bc26c62d5ef8cc.idx
.git/objects/pack/pack-6b3ea4129144c819898dd16a08bc26c62d5ef8cc.pack
```

Git± Packfiles

- “Tight Object Format”
- Opaque Format
    - Not Accessible
- Plumbing commands will still work

Using Git±

The porcelain over the pipes

- Plumbing commands are difficult, painful, and error prone
- Thankfully, we have friendly “porcelain” commands
- The basics can be covered with `git-add` and `git-commit`

git-init

- Create new local repository
- Better than manually creating a repository

git-init

```
% git init foobar
Initialized empty Git repository in /tmp/tmp.xbHJFvplCy/foobar/.git/
% cd foobar
± find .git
.git
.git/objects
.git/objects/info
.git/objects/pack
.git/config
.git/HEAD
.git/info
.git/info/exclude
.git/description
.git/hooks
.git/refs
.git/refs/tags
.git/refs/heads
```

git-status

- The go-to command for peering into the current state of a repository
- Provides information about state of all files
    - Currently untracked files
    - Currently modified files
    - Current state of “staging” area

git-add

Combines:

- `git-hash-object`
- `git-update-index`

git-add

```
% cd $(mktemp -d); git init bar; cd bar
± echo bar > bar.txt
± git add bar.txt
± git status
On branch trunk

Initial commit

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

        new file:   bar.txt
```

`git-commit`

- Creates a “commit” out of the current staging area
  - Requires a short message
  - Will implicitly figure out the parent commit
  - Forwards the `HEAD` pointer and the current branch pointer

git-commit

```
± git commit -m "Initial commit"
[trunk (root-commit) 8cbc334] Initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 bar.txt
± git status
On branch trunk
nothing to commit, working directory clean
```

`git-mv`

- Combines:
  - `mv`
  - `git-add`
- Rename detection is automatic

git-mv

```
± git mv foo bar.txt
On branch trunk
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    foo -> bar.txt
```

git-rm

- Removes the file from working tree
- Stops tracking the file
- Adds the removal to the staging area

git-rm

```
± git rm bar.txt
rm 'bar.txt'
± git status
On branch trunk
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    bar.txt

± git commit -m "remove bar.txt"
[trunk 8cbc334] remove bar.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 bar.txt
```

git-rm

- Can also use regular `rm` and `git-add`

Git± Branches

Git± Branches

- Named reference to a commit hash

Git± Branches

- Named reference to a commit hash
- Defined in text files under `./.git/refs`

Git± Branches

```
± find .git/refs -type f
.git/refs/heads/trunk
± cat .git/refs/heads/trunk
8cbc33461c800c20acea3b055886f8bed21c2092
```

git-branch

Branches can be created with a simple invocation of `git-branch`:

```
± git branch my_new_branch
± find .git/refs -type f
.git/refs/heads/my_new_branch
.git/refs/heads/trunk
```

git-checkout

After the branch is created, we can switch into that branch with `git-checkout`:

```
± git checkout my_new_branch
Switched to branch 'my_new_branch'
```

git-checkout -b

Or, we can do all in the same command:

```
± git checkout -b my_other_branch trunk
Switched to a new branch 'my_other_branch'
± find .git/refs -type f
.git/refs/heads/my_other_branch
.git/refs/heads/my_new_branch
.git/refs/heads/trunk
```

Git± Branches

Currently, all branches are pointing to the same commit:

```
± find .git/refs -type f -exec cat {} \;
8cbc33461c800c20acea3b055886f8bed21c2092
8cbc33461c800c20acea3b055886f8bed21c2092
8cbc33461c800c20acea3b055886f8bed21c2092
```

Git± Branches

```
± git branch
my_other_branch
± echo 'foo' > foo.txt
± git add foo.txt
± git commit -m 'add foo.txt'
[my_other_branch 3a8b37d] add foo.txt
 1 file changed, 1 insertion(+)
 create mode 100644 foo.txt
```

Git± Branches

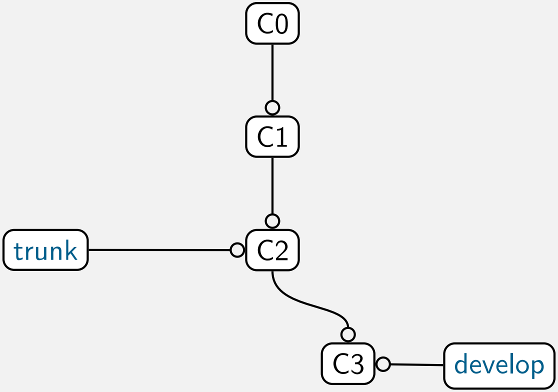
```
± find .git/refs -type -f
.git/refs/heads/my_other_branch
.git/refs/heads/my_new_branch
.git/refs/heads/trunk
± find .git/refs -type -f -exec cat {} \;
3a8b37dc0f79859b5b58f5cb0a859d4ddd0f99a0
8cbc33461c800c20acea3b055886f8bed21c2092
8cbc33461c800c20acea3b055886f8bed21c2092
```

Git± Merging

- Fast-Forward Merge
- *N*-parent Merge, where *N* usually is 2
- Both achieved via the `git-merge` command

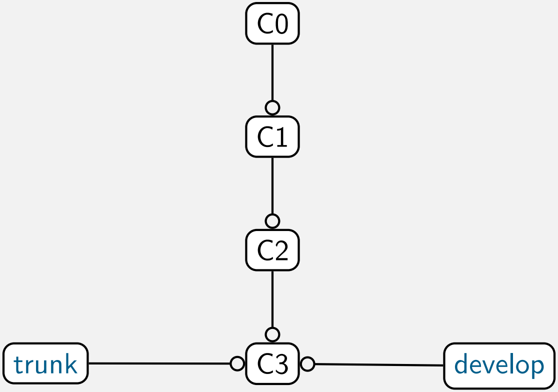
Git± Merging

Fast-Forward Merges



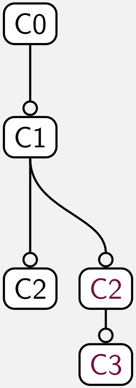
Git± Merging

Fast-Forward Merges



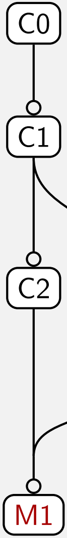
Git± Merging

2-Parent Merge



Git± Merging

2-Parent Merge



Working with remotes

- Clone a repository from, say, Github, will create the remote reference
- Otherwise, can be created with `git-remote`

Example Usage:

```
± git remote add origin ssh://remote_host/project_path
```

Working with remotes

- `git-clone`: Clone "remote" repository
  - `SSH://`: Bi-directional
  - `Git://`: Pull only, not authenticated
  - `HTTP (S)://`: Bi-directional, authenticated, unintelligent
  - `File://`: Strange
- `git-push`: Push local changes to remote repository
- `git-remote`: Utility command for working with remotes
- `git-pull`: Pull remote changes into working copy

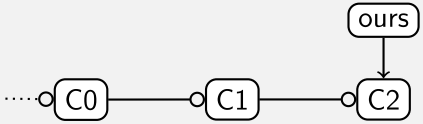
Git± Rebase

Another way to ~~merge~~ rewrite history

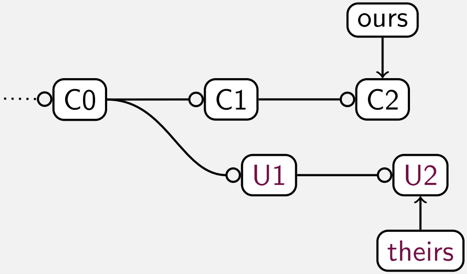
Or, more realistically, a great way to have a bad time...

- Merges branches via a pop, play, replay strategy
  1. Find common ancestor
  2. Pop “ours” off the ancestor
  3. Play “theirs” onto ancestor
  4. Replay “ours” onto result
- Inherently changes the replayed commits
- If this sounds scary, it is

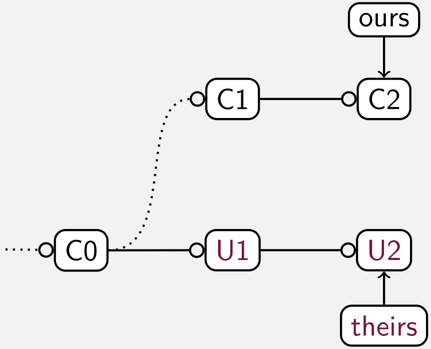
Git± Rebase



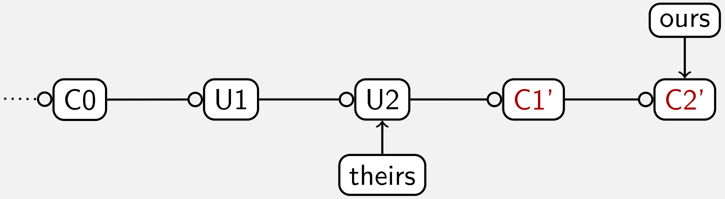
Git± Rebase



Git± Rebase



Git± Rebase



Git± Rebase

- When to rebase?
  - Squashing Work in Progress (WIP) commits
  - Rebase Private Trees
  - Pulling upstream changes in before sharing a new branch
  - Preference toward linear history
- When **not** to rebase?
  - Non-Private Tree
  - Other people's history (commits, usually)
  - Preference toward chronological history

Commit Frequency

How often to create commits

- WIP commits
- Logical Changes
    - Implemented a new feature
    - Fixed a bug

Writing Good Commit Messages

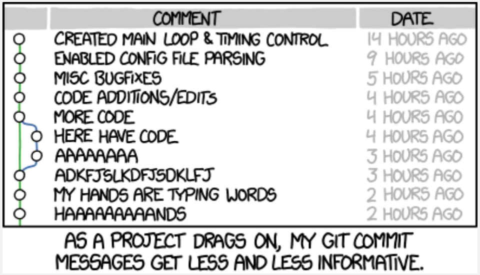


Figure: XKCD and Git Log[9]

Writing Good Commit Messages

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative voice in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain what and why vs. how

Writing Good Commit Messages

Why?

Because `git-log` output *needs* to be beautiful:

```
± git log --oneline | head -3
19d1c94 Remove unmerged files on :Gstatus U
9315ec6 Document StageUndo key map (U) in :Gstatus
9025078 Call git clean for U on untracked file
```

Writing Good Commit Messages

Examples

```
i dont think this stuff is needed
```

Writing Good Commit Messages

Examples

```
Convert ROM read access enable/disable string parsing to use the
`kstrtobool` function.

This fixes Bugzilla Bug 111301 -- Sysfs PCI rom file functionality does
not match documentation.

bugzilla: https://bugzilla.kernel.org/show_bug.cgi?id=111301

Reported-by: googlegot@xxxxxxxxx
Signed-off-by: Kenny Ballou <kballou@xxxxxxxxxxxxxx>
```

Writing Good Commit Messages
Examples

```
object.h: update flag allocation comment

Since the "flags" is shared, it's a good idea to keep track of who
uses what bit. When we need to use more flags in library code, we can
be sure it won't be re-used for another purpose by some caller.

While at there, fix the location of "5" (should be in a different
column than "4" two lines down)

Signed-off-by: Nguyễn Thái Ngọc Duy <pclouds@gmail.com>
Signed-off-by: Junio C Hamano <gitster@pobox.com>
```

Writing Good Commit Messages

- Be consistent

git-pull considered harmful

- Standard use of `git-pull` requires clean working directory
- Will force a merge, if drift between remote and local
- From the Git documentation [4], "Do not use `git pull` unless you actually want to merge the remote branch."
- I personally prefer using `git-fetch` and `git-merge`
- Another option: use `--ff-only` when pulling

```
[pull]
        ff = only
```

Git± Moving Forward

Git± Moving Forward

- Read the output

Git± Moving Forward

- Read the output
- No, *really*, Read the output!

Git± Moving Forward

- Read the output
- No, *really*, Read the output!
- `git-scm.com` and “Pro Git”

Git± Moving Forward

- Read the output
- No, *really*, Read the output!
- `git-scm.com` and “Pro Git”
- `#git` on Libre.Chat

Git± Moving Forward

- Read the output
- No, *really*, Read the output!
- `git-scm.com` and “Pro Git”
- `#git` on Libre.Chat
- Git± man pages [3] [6]

Git± Moving Forward

- Read the output
- No, *really*, Read the output!
- `git-scm.com` and “Pro Git”
- `#git` on Libre.Chat
- Git± man pages [3] [6]
- Git± workflows [4]

Learning Git± in Reverse

A Backwards Introduction to the "information manager from hell" [e83c51633]

Kenny Ballou

California State University San Marcos

Spring 2025

References I

[1] Chris Beams. *Git Commit Messages*. 2017. URL: http://chris.beams.io/posts/git-commit/.

[2] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014.

[3] Git Contributors. *Git Man Pages*. 2017. URL: https://www.kernel.org/pub/software/scm/git/docs/.

[4] Linux Contributors. *Git Workflows*. 2017. URL: https://www.kernel.org/pub/software/scm/git/docs/gitworkflows.html.

[5] Various Contributors. *Git-SCM Site*. 2017. URL: http://git-scm.com/.

[6] Ian Miel. *Five Key Git Concepts Explained the Hard Way*. 2017. URL: https://zwischenzugs.com/2018/03/14/five-key-git-concepts-explained-the-hard-way/.

References II

[7] Aditya Mukerjee. *Unpacking Git Packfiles*. 2017. URL: https://codewords.recurse.com/issues/three/unpacking-git-packfiles/.

[8] Randall Munroe. *Git*. 2017. URL: https://xkcd.com/1597.

[9] Randall Munroe. *Git Commit*. 2017. URL: http://xkcd.com/1296/.

[10] Tim Pope. *A Note About Git Commit Messages*. 2017. URL: http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html.

[11] StackOverflow, ed. *Stack Overflow: Git Pull Considered Harmful*. 2017. URL: http://stackoverflow.com/questions/15316601/in-what-cases-could-git-pull-be-harmful#15316602.

References III

[12] Linus Torvalds. *e83c51633*. 2007. URL: `https://github.com/git/git/commit/e83c5163316f89bfbde7d9ab23ca2e25604af290`.

[13] Linus Torvalds. *Linus on Rebasing*. 2017. URL: `http://www.mail-archive.com/dri-devel@lists.sourceforge.net/msg39091.html`.

[14] Wikipedia, ed. *zlib*. 2017. URL: `https://en.wikipedia.org/wiki/Zlib`.