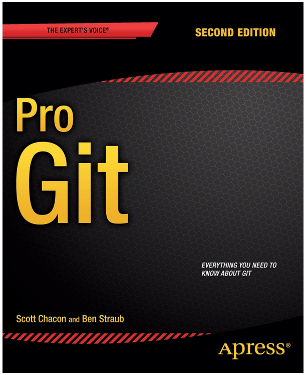
Git

Crawford Kesy Moderegger

January 25, 2025

Literature

Title: Pro Git
Author: Scott Chacon and Ben Straub
Edition: 2nd
Year: 2014
Publisher: Apress
URL: `https://git-scm.com/book/en/v2`


THE EXPERT'S VOICE®
SECOND EDITION
Pro Git
EVERYTHING YOU NEED TO KNOW ABOUT GIT
Scott Chacon and Ben Straub
Apress®

Version Control Systems

- What is a Version Control System (VCS)?
    - Tracks changes to files

Version Control Systems

- What is a Version Control System (VCS)?
  - Tracks changes to files
- Centralized vs. Distributed VCS
  - Centralized: All clients depend on a single server
  - Distributed: Every client has a full copy of the repository

Version Control Systems

- What is a Version Control System (VCS)?
    - Tracks changes to files
- Centralized vs. Distributed VCS
    - Centralized: All clients depend on a single server
    - Distributed: Every client has a full copy of the repository
- Advantages of a VCS
    - Recover lost or overwritten data by reverting to previous versions
    - Enables collaboration

Git

- A Distributed VCS

The Git logo

Git

- A Distributed VCS
- Created by Linus Torvalds in 2005

The Git logo

Git

- A Distributed VCS
- Created by Linus Torvalds in 2005
- Creates/stores snapshots of your project

The Git logo

Git

- A Distributed VCS
- Created by Linus Torvalds in 2005
- Creates/stores snapshots of your project
- Goals of Git
  - Speed
  - Simple design
  - Strong support for non-linear development
  - Fully distributed
  - Able to handle large projects

The Git logo

Git Basics - Get a Repository

- To create a new repository, run:

```
$ git init
```

Git Basics - Get a Repository

- To create a new repository, run:

```
$ git init
```

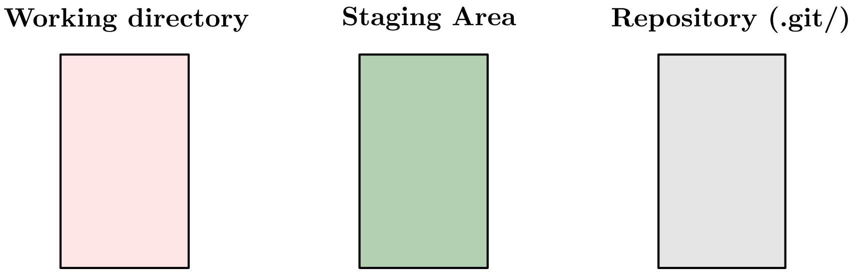
- This creates a new .git/ directory in the root of your project
  - Stores all the data and metadata Git needs to manage the repository
  - Losing or corrupting this directory means losing the repository's version history

```
.git/
  | config
  | description
  | HEAD
  | hooks/
  | info/
  | objects/
  | refs/
```

Newly created Git directory

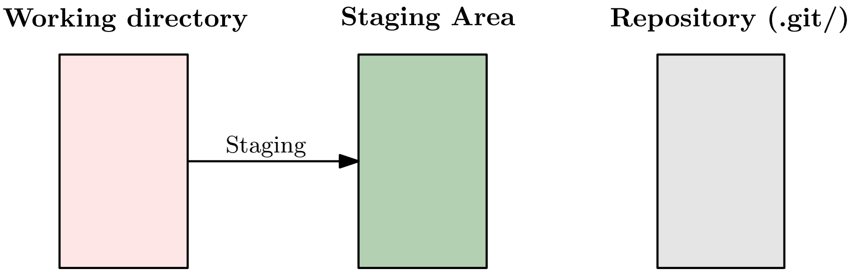
Git Workflow

- A basic Git workflow looks like this:



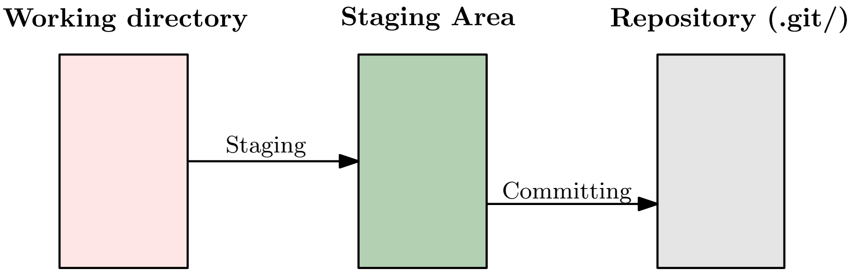
Git Workflow

- A basic Git workflow looks like this:



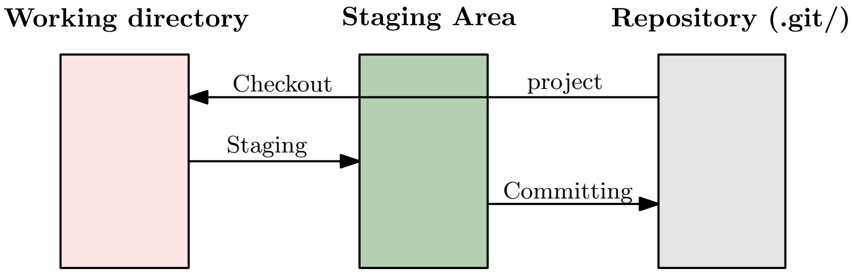
Git Workflow

- A basic Git workflow looks like this:

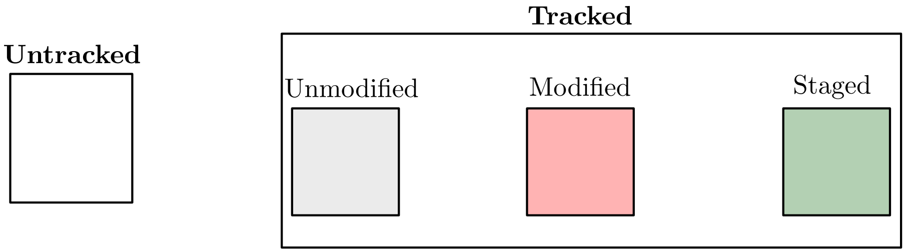


Git Workflow

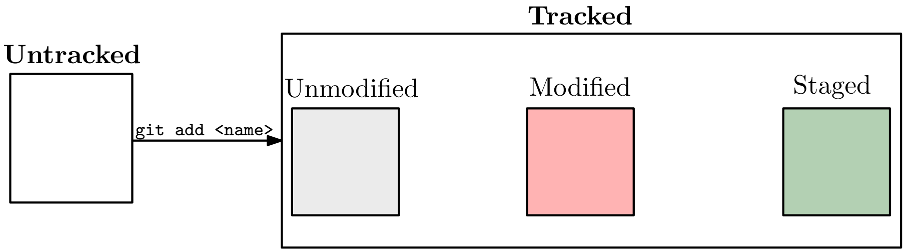
- A basic Git workflow looks like this:



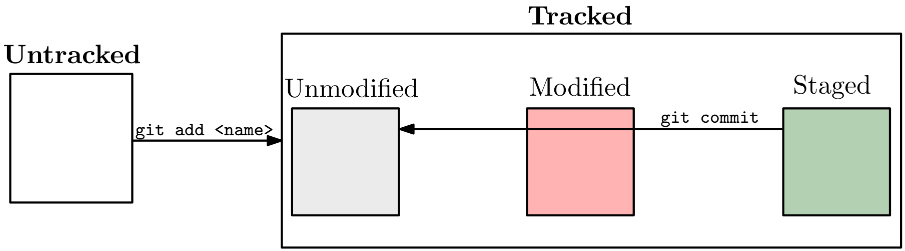
The different File States in Git



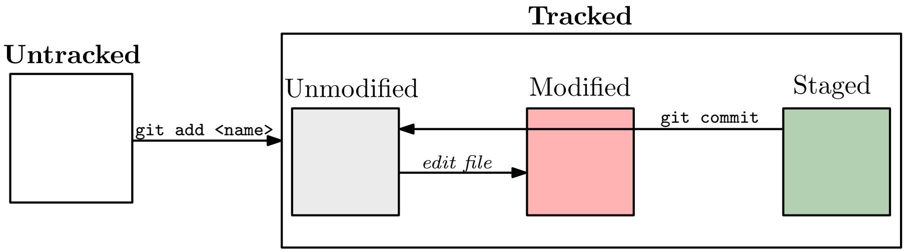
The different File States in Git



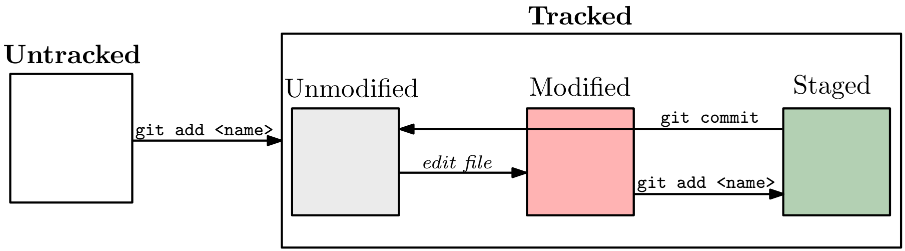
The different File States in Git



The different File States in Git



The different File States in Git



Git Objects

- Objects make up most of the data stored in the repository

Git Objects

- Objects make up most of the data stored in the repository
- They are referenced by a hash

Git Objects

- Objects make up most of the data stored in the repository
- They are referenced by a hash
- There are three main types of objects
    1. Blob

Git Objects

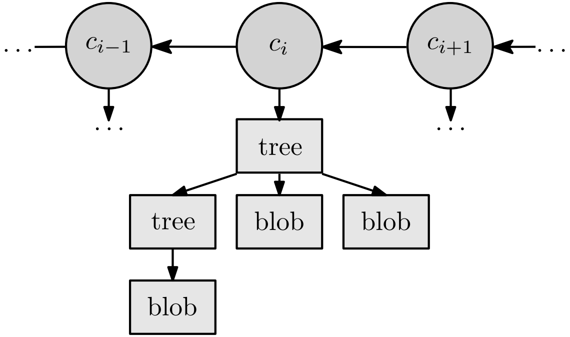
- Objects make up most of the data stored in the repository
- They are referenced by a hash
- There are three main types of objects
    1. Blob
    2. Tree

Git Objects

- Objects make up most of the data stored in the repository
- They are referenced by a hash
- There are three main types of objects
  1. Blob
  2. Tree
  3. Commit

Git Objects

- Objects make up most of the data stored in the repository
- They are referenced by a hash
- There are three main types of objects
  1. Blob
  2. Tree
  3. Commit



Object hierarchy

Git Index

- Also referred to as ”Staging Area” or ”Cache”

Git Index

- Also referred to as "Staging Area" or "Cache"
- Stored as binary at the root of `.git/`

Git Index

- Also referred to as "Staging Area" or "Cache"
- Stored as binary at the root of `.git/`
- Records a sorted list of path names with corresponding references to objects

Git Index

- Also referred to as "Staging Area" or "Cache"
- Stored as binary at the root of `.git/`
- Records a sorted list of path names with corresponding references to objects
- `git add <file>`
  1. Creates a new blob for the contents of `<file>`
  2. Adds the path and the blob's hash to the index

Git Index

- Also referred to as "Staging Area" or "Cache"
- Stored as binary at the root of `.git/`
- Records a sorted list of path names with corresponding references to objects

- `git add <file>`
  1. Creates a new blob for the contents of `<file>`
  2. Adds the path and the blob's hash to the index

- `git commit`
  1. Writes a new tree object based on the contents in the index
  2. The new commit object points to this tree

Git Branches

- Enables parallel development with little risk

Git Branches

- Enables parallel development with little risk
- A branch is a pointer to a commit

Git Branches

- Enables parallel development with little risk
- A branch is a pointer to a commit
- `HEAD`
  - Pointer to (usually) a branch to keep track of current branch
  - A new commit moves the branch pointer, that `HEAD` points to, to the new object

Git Branches

- Enables parallel development with little risk
- A branch is a pointer to a commit
- `HEAD`
  - Pointer to (usually) a branch to keep track of current branch
  - A new commit moves the branch pointer, that `HEAD` points to, to the new object
- To create a new branch:

```
$ git branch <name>
```

Git Branches

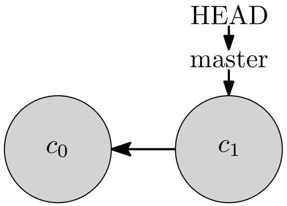
- Enables parallel development with little risk
- A branch is a pointer to a commit
- `HEAD`
  - Pointer to (usually) a branch to keep track of current branch
  - A new commit moves the branch pointer, that `HEAD` points to, to the new object
- To create a new branch:

```
$ git branch <name>
```

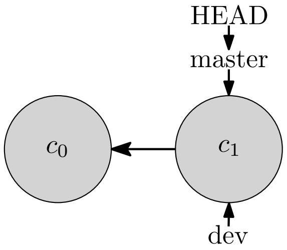
- To move `HEAD` to another branch (switching branches):

```
$ git checkout <branch>
```

Git Branches - An example

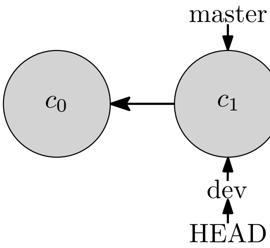


Git Branches - An example



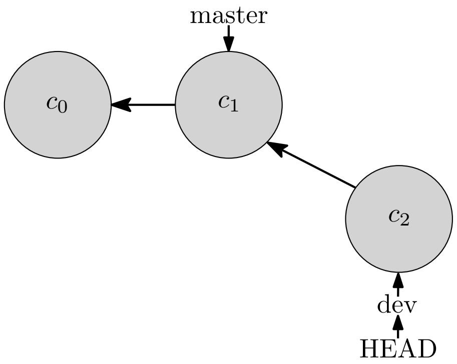
```
$ git branch dev
```

Git Branches - An example



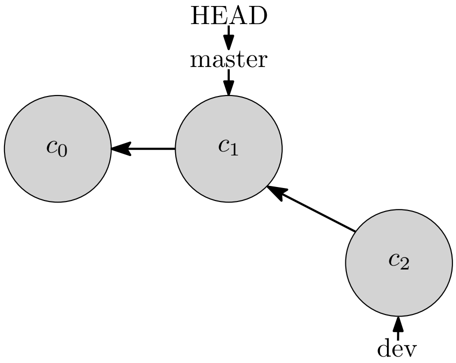
```
$ git checkout dev
```

Git Branches - An example



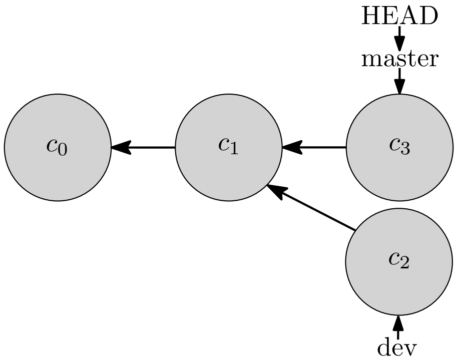
```
...
$ git commit
```

Git Branches - An example



```
$ git checkout master
```

Git Branches - An example



```
...
$ git commit
```

Merging branches

- Reconciling two branches is done by merging one into the other

Merging branches

- Reconciling two branches is done by merging one into the other

```
$ git merge <branch>
```

- This will merge `<branch>` into `HEAD`'s branch

Merging branches

- Reconciling two branches is done by merging one into the other

```
$ git merge <branch>
```

- This will merge `<branch>` into `HEAD`'s branch

- A merge most commonly uses a 3-way merge algorithm:
  1. The common ancestor is used as the base

Merging branches

- Reconciling two branches is done by merging one into the other

```
$ git merge <branch>
```

- This will merge `<branch>` into `HEAD`'s branch

- A merge most commonly uses a 3-way merge algorithm:
  1. The common ancestor is used as the base
  2. The tips of the branches are compared with the base to determine the new changes on each branch

Merging branches

- Reconciling two branches is done by merging one into the other

```
$ git merge <branch>
```

- This will merge `<branch>` into `HEAD`'s branch

- A merge most commonly uses a 3-way merge algorithm:
  1. The common ancestor is used as the base
  2. The tips of the branches are compared with the base to determine the new changes on each branch
  3. Non-conflicting changes are applied

Merging branches

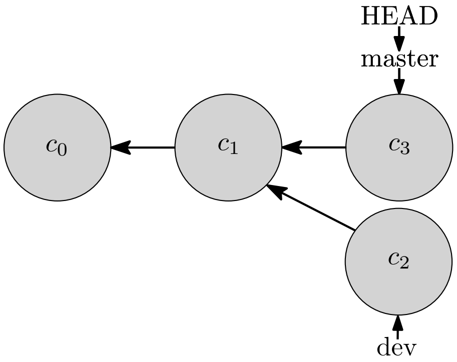
- Reconciling two branches is done by merging one into the other

```
$ git merge <branch>
```

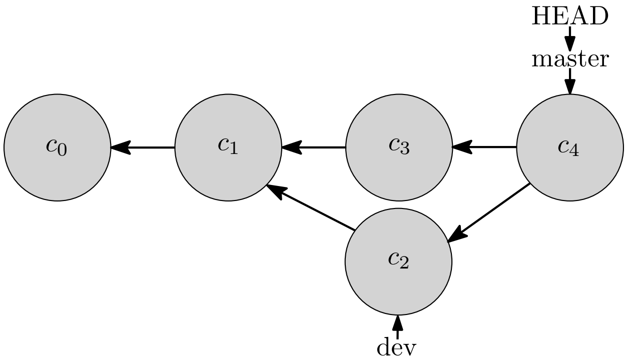
- This will merge `<branch>` into `HEAD`'s branch

- A merge most commonly uses a 3-way merge algorithm:
  1. The common ancestor is used as the base
  2. The tips of the branches are compared with the base to determine the new changes on each branch
  3. Non-conflicting changes are applied
  4. Conflicting changes result in a merge conflict, which must be resolved manually

True Merge

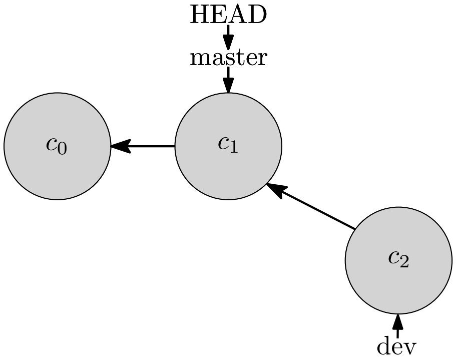


True Merge

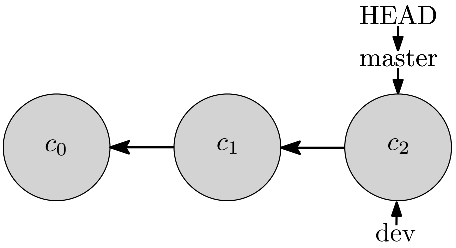


```
$ git merge dev
```

Fast-forward Merge



Fast-forward Merge



```
$ git merge dev
```

Git References

- References are pointers to a certain commit

Git References

- References are pointers to a certain commit
- They are stored in `.git/refs/`

```
.git/refs/
  | heads/
  |   | master
  |   | dev
  | remotes/
```

```
$ cat .git/refs/heads/master
98c7705f...
$ git log master | head -1
commit 98c7705f...
```

Git References

- References are pointers to a certain commit
- They are stored in `.git/refs/`

```
.git/refs/
  | heads/
  |   | master
  |   | dev
  | remotes/
```

```
$ cat .git/refs/heads/master
98c7705f...
$ git log master | head -1
commit 98c7705f...
```

- While `.git/HEAD` is not under `.git/refs/`, it is a symbolic reference to another reference

```
$ cat .git/HEAD
ref: refs/heads/master
```

Git Remotes

- References to other repositories hosted remotely

Git Remotes

- References to other repositories hosted remotely
- Remote references are used to represent references from the remote

Git Remotes

- References to other repositories hosted remotely
- Remote references are used to represent references from the remote
- For each branch `<branch>` from the remote, a remote reference is created
  - Denoted as `<remote>/<branch>`
  - Not directly accessible
  - Automatically updated
  - Stored under `.git/refs/remotes`

Git Remotes

- References to other repositories hosted remotely
- Remote references are used to represent references from the remote
- For each branch `<branch>` from the remote, a remote reference is created
  - Denoted as `<remote>/<branch>`
  - Not directly accessible
  - Automatically updated
  - Stored under `.git/refs/remotes`
- Adding a Remote:

```
$ git remote add <name> <url>
```

Git Remotes - Pushing and Pulling

- **Pushing** changes to a remote:

```
$ git push <remote> <branch>
```

  - This will push all changes from `<branch>` to the remote repository `<remote>`
  - Also updates the remote branch `<remote>/<branch>`

Git Remotes - Pushing and Pulling

- **Pushing** changes to a remote:

```
$ git push <remote> <branch>
```

  - This will push all changes from `<branch>` to the remote repository `<remote>`
  - Also updates the remote branch `<remote>/<branch>`

- **Pulling** in changes from a remote:

```
$ git fetch <remote>
```

  - This will download all changes and update all remote references

Git Remotes - Pushing and Pulling

- **Pushing** changes to a remote:

```
$ git push <remote> <branch>
```

  - This will push all changes from `<branch>` to the remote repository `<remote>`
  - Also updates the remote branch `<remote>/<branch>`

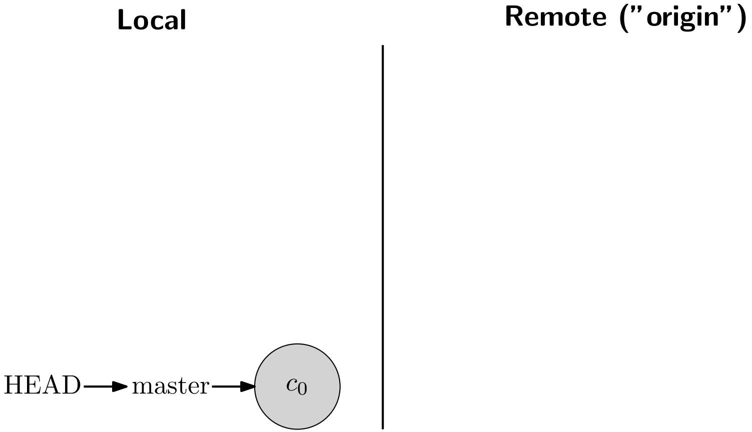
- **Pulling** in changes from a remote:

```
$ git fetch <remote>
```

  - This will download all changes and update all remote references
  - To integrate the changes, merge a remote branch into the current branch:

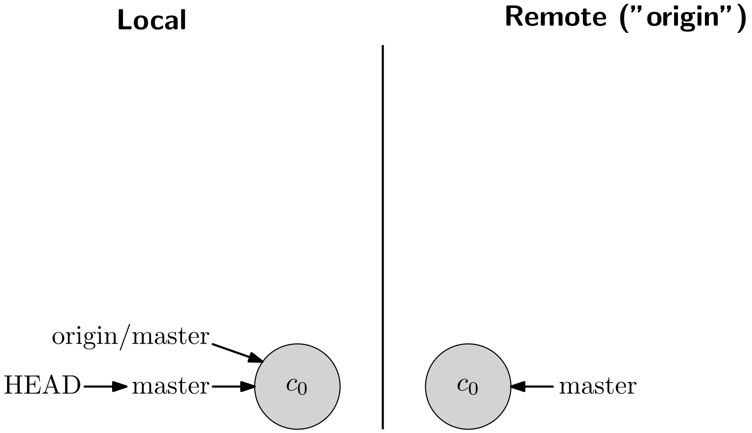
```
$ git merge <remote>/<branch>
```

Remotes - An Example



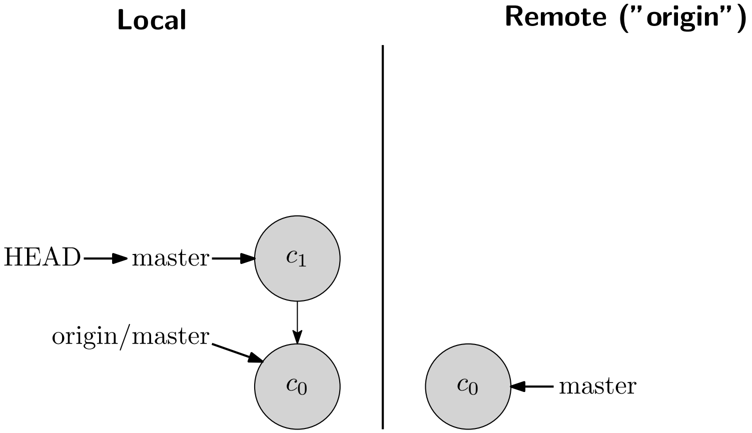
```
$ git remote add origin git@github.com:johnsmith/myrepo.git
```

Remotes - An Example



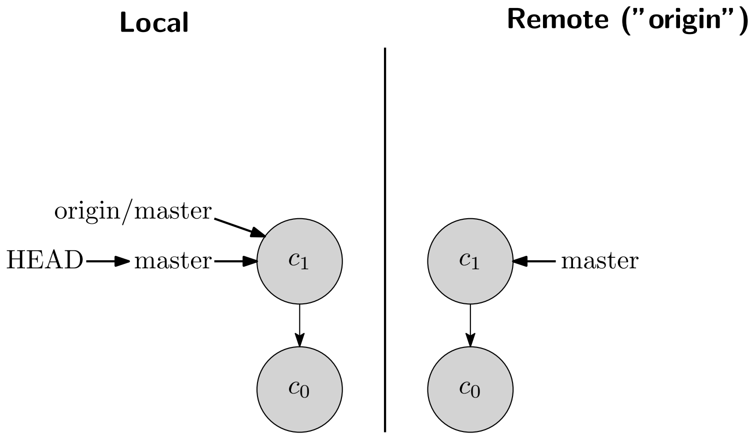
```
$ git push origin master
```

Remotes - An Example



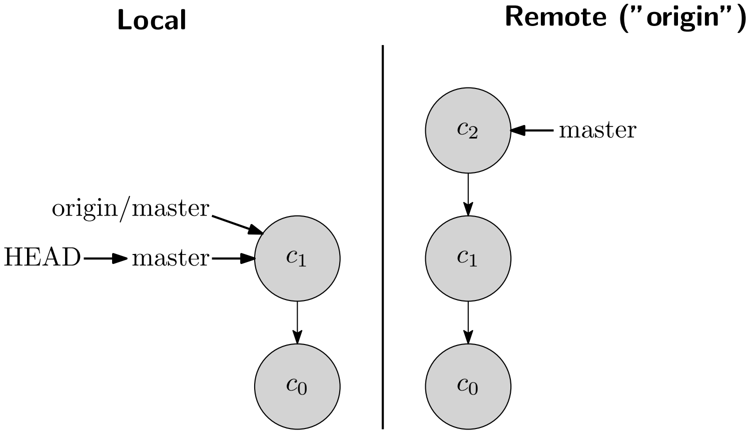
```
...
$ git commit
```

Remotes - An Example

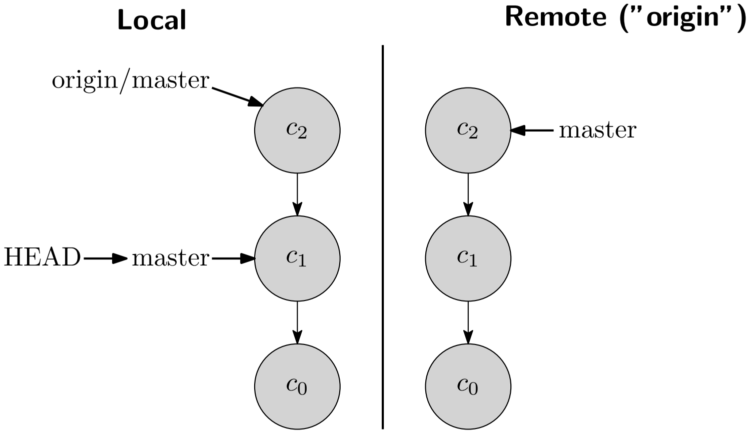


```
$ git push origin master
```

Remotes - An Example

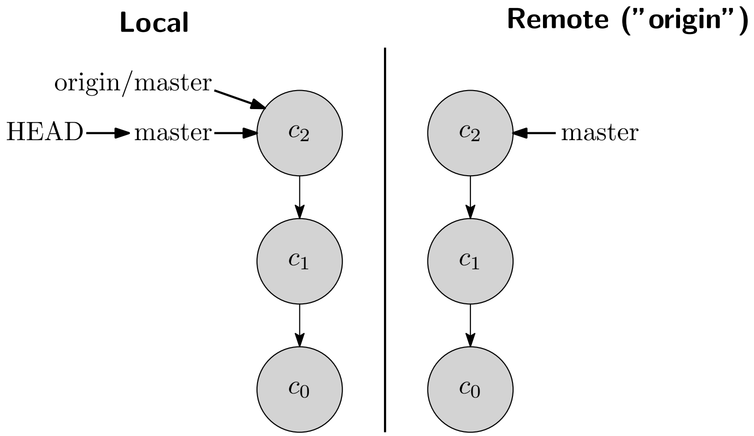


Remotes - An Example



```
$ git fetch origin
```

Remotes - An Example



```
$ git merge origin/master
```

Undo Operations

- Most operations in Git are reversible

Undo Operations

- Most operations in Git are reversible
- To undo uncommitted actions:

```
$ git restore <file>
```

Undo Operations

- Most operations in Git are reversible
- To undo uncommitted actions:

```
$ git restore <file>
```

- To undo commits:

```
$ git reset <commit>
```

  - Moves the current branch pointer to `<commit>`
  - Does not modify your working directory

Undo Operations

- Most operations in Git are reversible
- To undo uncommitted actions:

```
$ git restore <file>
```

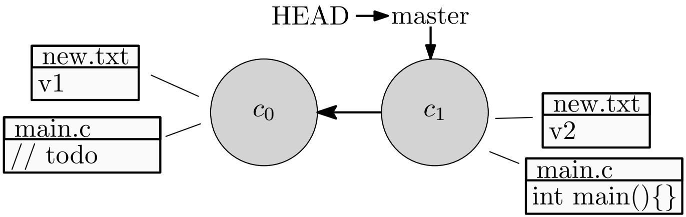
- To undo commits:

```
$ git reset <commit>
```

  - Moves the current branch pointer to `<commit>`
  - Does not modify your working directory
- `git reset` can cause partial loss of history

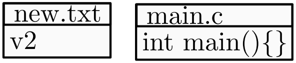
Undo Operations - An example

- **Goal:** Revert back to state of c_0, but keep contents of `main.c`



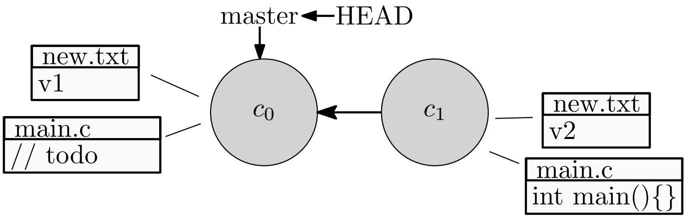
```
$ git log
commit 4b76761cd4...
...
commit 52cbc76923...
...
```

Working directory:



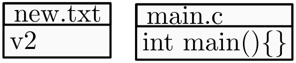
Undo Operations - An example

- **Goal:** Revert back to state of c_0, but keep contents of `main.c`



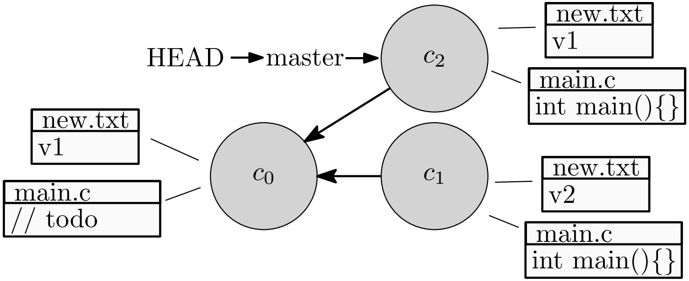
```
$ git reset 52cb
```

Working directory:



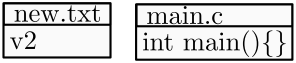
Undo Operations - An example

- **Goal:** Revert back to state of c_0, but keep contents of `main.c`



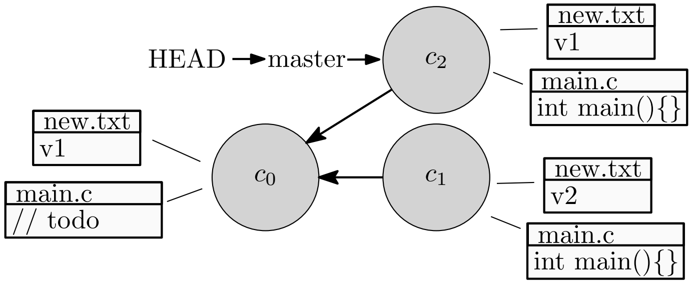
```
$ git add main.c
$ git commit
```

Working directory:



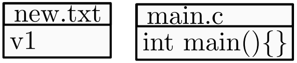
Undo Operations - An example

- **Goal:** Revert back to state of c_0, but keep contents of `main.c`



```
$ git restore new.txt
```

Working directory:



Git Optimizations

Git Optimizations

- **Compression:** Git uses zlib to compress objects

Git Optimizations

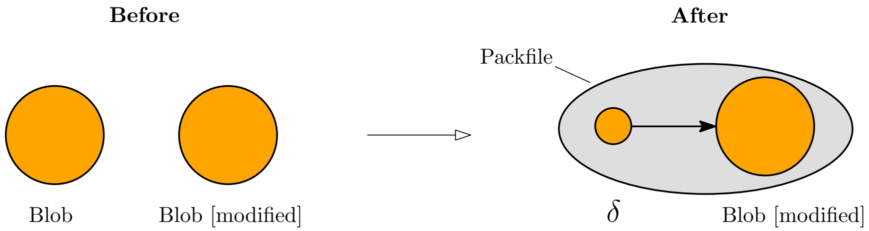
- **Compression:** Git uses zlib to compress objects
- **References:** Git uses references to avoid storing duplicates

Git Optimizations

- **Compression:** Git uses zlib to compress objects
- **References:** Git uses references to avoid storing duplicates
- **Packfiles:** Git occasionally creates so-called packfiles
  - "Loose" objects packed into one file
  - Also creates corresponding index-files for efficient access

Git Optimizations

- **Compression:** Git uses zlib to compress objects
- **References:** Git uses references to avoid storing duplicates
- **Packfiles:** Git occasionally creates so-called packfiles
    - "Loose" objects packed into one file
    - Also creates corresponding index-files for efficient access



Packfile example

The Git Directory - Summary

```
.git/
  | config
  | description
  | HEAD
  | index
  | hooks/
  | info/
  | logs/
  | objects/
  | refs/
```

Git directory

The Git Directory - Summary

- `config` and `description` are self-explanatory

```
.git/
   | config
   | description
   | HEAD
   | index
   | hooks/
   | info/
   | logs/
   | objects/
   | refs/
```

Git directory

The Git Directory - Summary

- config and description are self-explanatory
- HEAD: Reference to a branch

```
.git/
   | config
   | description
   | HEAD
   | index
   | hooks/
   | info/
   | logs/
   | objects/
   | refs/
```

Git directory

The Git Directory - Summary

- `config` and `description` are self-explanatory
- `HEAD`: Reference to a branch
- `index`: The staging area

```
.git/
  | config
  | description
  | HEAD
  | index
  | hooks/
  | info/
  | logs/
  | objects/
  | refs/
```

Git directory

The Git Directory - Summary

- `config` and `description` are self-explanatory
- `HEAD`: Reference to a branch
- `index`: The staging area
- `hooks/`: Scripts triggered by certain commands

```
.git/
   | config
   | description
   | HEAD
   | index
   | hooks/
   | info/
   | logs/
   | objects/
   | refs/
```

Git directory

The Git Directory - Summary

- `config` and `description` are self-explanatory
- `HEAD`: Reference to a branch
- `index`: The staging area
- `hooks/`: Scripts triggered by certain commands
- `info/`: Local configurations

```
.git/
  | config
  | description
  | HEAD
  | index
  | hooks/
  | info/
  | logs/
  | objects/
  | refs/
```

Git directory

The Git Directory - Summary

- `config` and `description` are self-explanatory
- `HEAD`: Reference to a branch
- `index`: The staging area
- `hooks/`: Scripts triggered by certain commands
- `info/`: Local configurations
- `logs/`: Records changes made to refs

```
.git/
   | config
   | description
   | HEAD
   | index
   | hooks/
   | info/
   | logs/
   | objects/
   | refs/
```

Git directory

The Git Directory - Summary

- `config` and `description` are self-explanatory
- `HEAD`: Reference to a branch
- `index`: The staging area
- `hooks/`: Scripts triggered by certain commands
- `info/`: Local configurations
- `logs/`: Records changes made to refs
- `objects/`: All objects

```
.git/
  | config
  | description
  | HEAD
  | index
  | hooks/
  | info/
  | logs/
  | objects/
  | refs/
```

Git directory

The Git Directory - Summary

- config and description are self-explanatory
- HEAD: Reference to a branch
- index: The staging area
- hooks/: Scripts triggered by certain commands
- info/: Local configurations
- logs/: Records changes made to refs
- objects/: All objects
- refs/: (Remote) branches

```
.git/
   | config
   | description
   | HEAD
   | index
   | hooks/
   | info/
   | logs/
   | objects/
   | refs/
```

Git directory