GIT

Amitabha Sanyal

Acknowledgements

Many figures in these slides have been taken from the following book, that is also available under a Creative Commons Attribution Non-Commercial Share Alike 3.0 license.

- Pro Git book, written by Scott Chacon and Ben Straub.

Here is the link to the book made available by the authors.

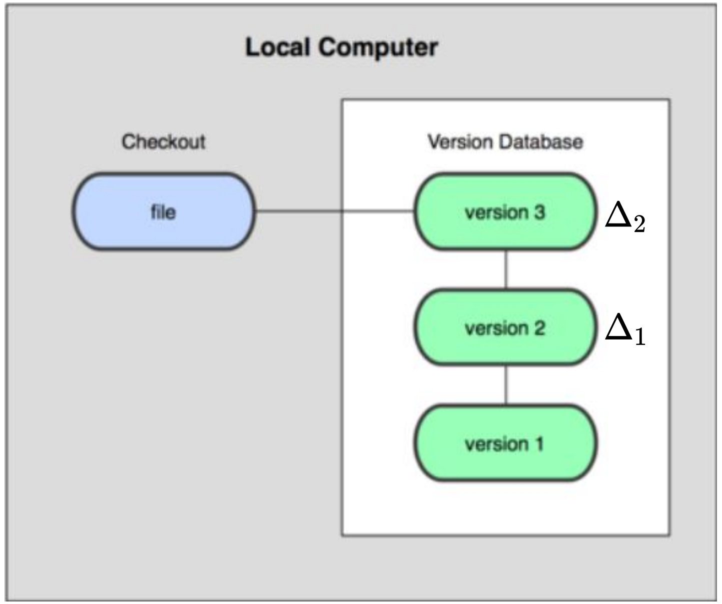
Outline

What is a version control system?

- Software is built
  - Incrementally
  - In collaboration
  - As more than one independent strands of development.
- **Version control** is a system that records changes to a file or set of files over time so that you can recall specific versions later.

> - Checkout - the version that you are currently working on
> - Patch set - The difference Δ between one version another

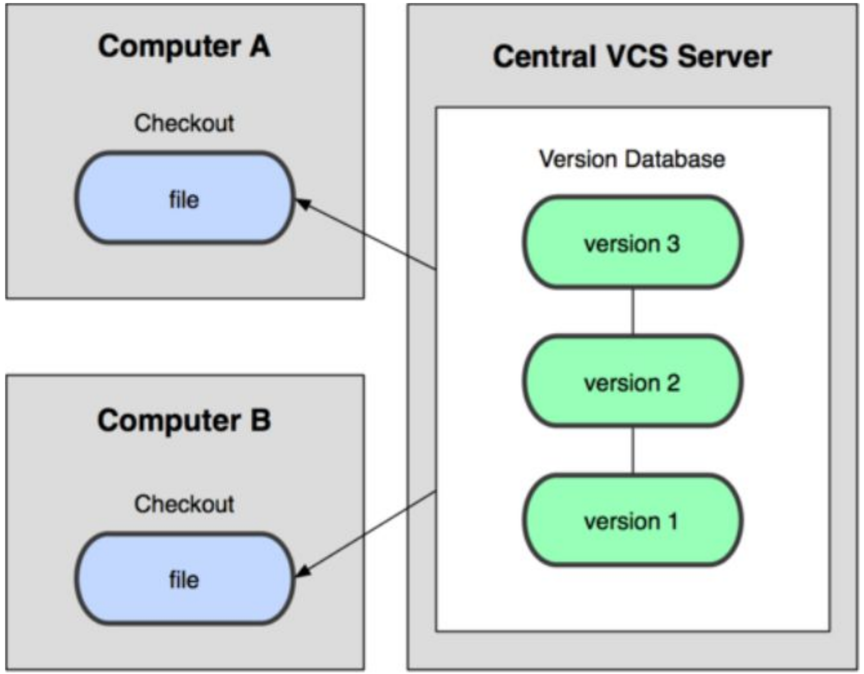
A local version control system (rcs)



What is a version control system?

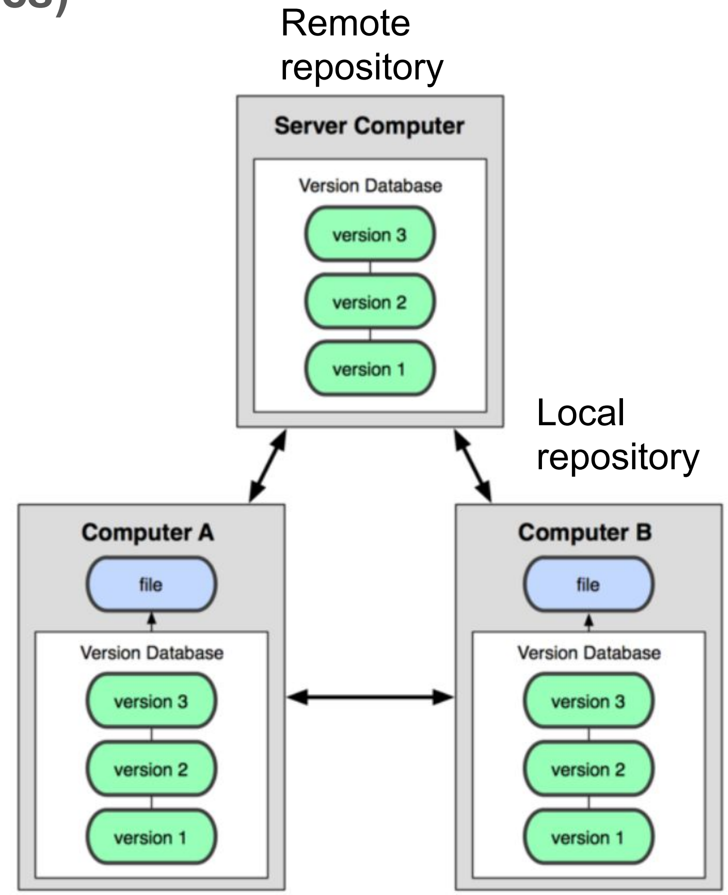
- **A centralized version control system (cvs, svn)**

  - Possibility of collaboration.
  - Centralized server is vulnerable.



What is a version control system?

- **A distributed version control system (git, Darcs)**
  - Each client fully mirrors the repository.
  - Vulnerability of centralized server is minimized.
  - If the server dies, any of the clients can be copied back.



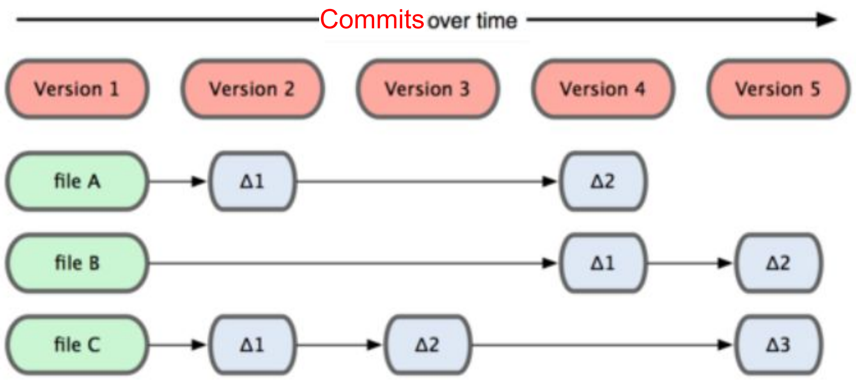
git

- **Design goals:**
  - Simple design
  - Speed
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed

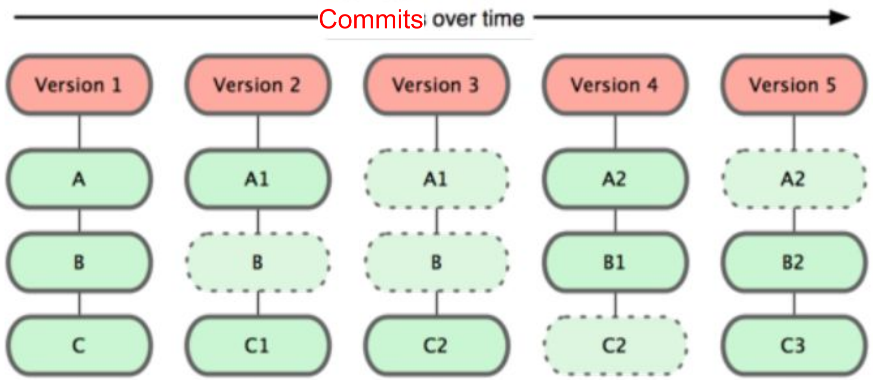
Able to handle large projects like the Linux kernel efficiently (speed and data size)

git does not store patches

other vc systems



git

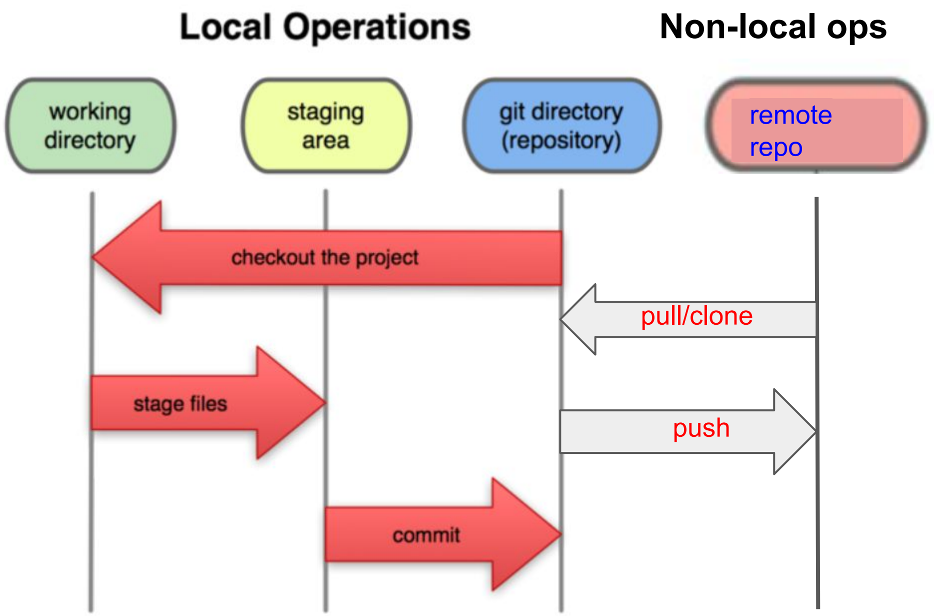


- Every commit is a reference to a full record of all the files.
- Most operations are local.
- Git has integrity. Everything in git is SHA-1 hashed. A file or a directory is referred by the hash value, such as

Example hash value: 24b9da6552252987aa493b52f8696cd6d3b00373

git - the larger picture

- Basic workflow of local operations
    - Assume a version in your working directory
    - Make changes to the files in working directory
    - Stage some or all of the modified files.
    - Commit to local repo.
- Non-local operations later.



Outline

1. Introduction to Version Control Systems and git
2. Setting up git and basic commands

Setting up git

```
>  sudo apt-get install git
>  git config --global user.name "<user name>"
>  git config --global user.email <email address>
>  git config --global core.editor emacs
>  git config --global merge.tool meld
>  git config --list

   user.email=<email address>
   user.name=<user name>
   core.editor=emacs
   merge.tool=meld
   ...

>  mkdir demo-cS251-2020; cd demo-cs251-2020
>  git init
```

got - basic commands

```
 >  touch file1.txt      # create file1.txt
 >  touch file2.txt      # create file2.txt
 >  git add file1.txt    # stage  file1.txt
 >  git status           # show the state of git
 >

On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
   new file:   file1.txt
Untracked files:
  (use "git add <file>..." to include in what will be committed)
   file2.txt
```

git - basic commands

```
 >  touch file1.txt      # create file1.txt
 >  touch file2.txt      # create file2.txt
 >  git add file1.txt    # stage  file1.txt
 >  git status           # show the state of git

On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
   new file:   file1.txt
Untracked files:
  (use "git add <file>..." to include in what will be committed)
   file2.txt
```

git - basic commands

```
>  git commit file1.txt -m"Committting the file file1.txt"
   [master (root-commit) 91bb7e6] Committting the file file1.txt
   1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 file1.txt
>  git status              # show the state of git
   On branch master
   Untracked files:
     (use "git add <file>..." to include in what will be committed)
     file2.txt
>  git log
   commit 91bb7e6712ba65786976a7d23d88b0a8269b6044 (HEAD -> master)
   Author: Amitabha Sanyal <amit23358@gmail.com>
   Date:   Tue Aug 18 21:16:18 2020 +0530

    Committting the file file1.txt
```

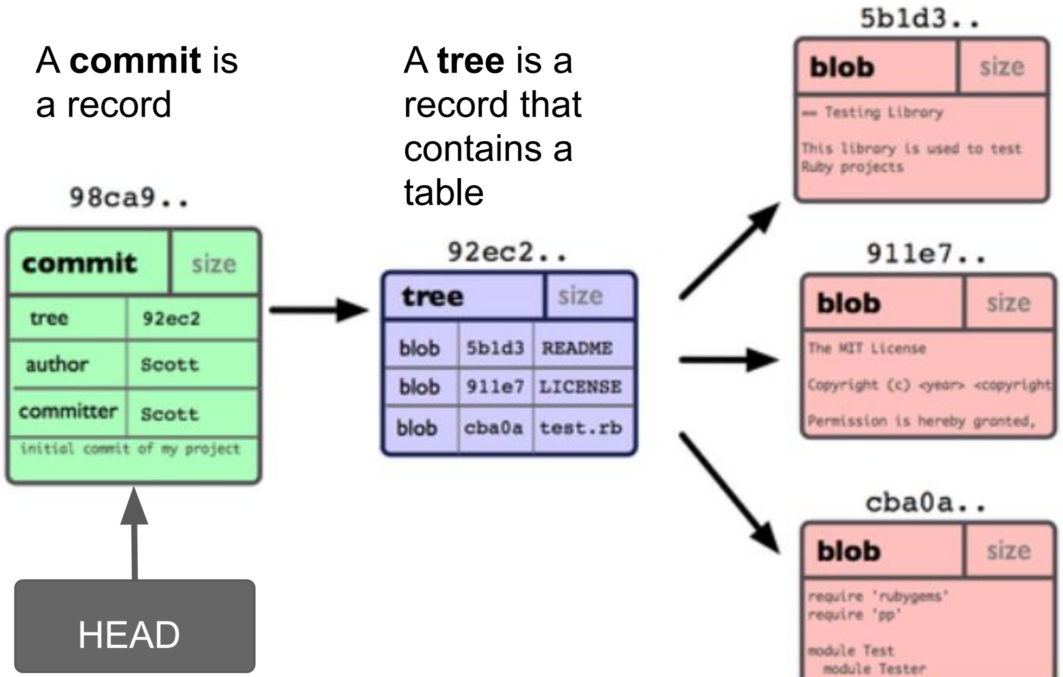
Git objects and their structures

- Git objects consist of **commits**, **trees** and **blobs**
- Each object is named by its hash value.

A **commit** is a record

A **tree** is a record that contains a table

A **blob** is the actual content of the file.



Three areas of GIT

- `create file.txt`

Working area	Staging area	Commit
`file.txt - v1`		

- `git add file.txt`

Working area	Staging area	Commit
`file.txt - v1`	`file.txt - v1`	

- `git commit -m "msg"`

Working area	Staging area	Commit
`file.txt - v1`	`file.txt - v1`	`file.txt - v1`

- `edit file.txt`

Working area	Staging area	Commit
`file.txt - v2`	`file.txt - v1`	`file.txt - v1`

States of a file

- add `file.txt`

Working area	Staging area	Commit
`file.txt - v2`	`file.txt - v2`	`file.txt - v1`

- `edit file.txt`

Working area	Staging area	Commit
`file.txt - v3`	`file.txt - v2`	`file.txt - v1`

- `git commit -m "msg"`
  `git commit file.txt -m "msg"`

Working area	Staging area	Commit
`file.txt - v3`	`file.txt - v2`	`file.txt - v2`
`file.txt - v3`	`file.txt - v3`	`file.txt - v3`

`git diff`

- After a commit, a file may be changed. It can be in:
  - a modified state, or
  - a staged state
- `diff` gives the changes after the last commit
  - `git diff`: differences between commit and modified file
  - `git diff --cached`: differences between commit and staged file
- Notation:

  `@@ -1,4 +1,6 @@`

  `4` lines starting from line `1` in the original file changed to `6` lines starting from line 1 in the changed file.
- Following this the changed lines are shown:

  + line means line added

  - line means line deleted

  Example follows.

git diff --cached

`git show HEAD:file2.txt`

```
A line has been added
A second line has been added.
 A third line
A fourth line
```

`cat file2.txt`

```
To illustrate diff adding a
new line
A line has been added
A second line has been added.
 A third line
 And another line here
A fourth line
```

```
>  git diff  --cached
   ...
   @@ -1,4 +1,6 @@
   +To illustrate diff adding a new line --Added line
    A line has been added               -- line not changed
    A second line has been added.       -- line not changed
     A third line                       -- line not changed
   + And another line here              --Added line
    A fourth line                       -- line not changed
```

The log of all commits-- `git log`

- `git log -p` reports successive commits and their diffs

```
commit 42ad8670a857f9cc3392ef5678cfb02684052274 (HEAD -> master)
Author: Amitabha Sanyal <amit23358@gmail.com>
...
diff --git a/file2.txt b/file2.txt
index c2fabbc..199f994 100644
--- a/file2.txt
+++ b/file2.txt
@@ -1,4 +1,6 @@
+To illustrate diff adding a new line
 A line has been added
 A second line has been added.
  A third line
+ And another line here
 A fourth line

commit 7acf879c379ea57e394f322159c152f184421313
Author: Amitabha Sanyal <amit23358@gmail.com>
...
```

Commands that provide information

- **`git cat-file -p object`**
  Example: `git cat-file -p HEAD`
  Displays the commit object `HEAD`.

  `Example: git cat-file -p 383e560d` (`383e560d` is a file object)
  Displays the file object `383e560d`

- **`git ls-tree object`**
  Example: git ls-tree HEAD. Prints the tree component of **`HEAD`**.

- **`git ls-files -s`**
  shows the files in the staging area

Commands that provide information

> **`git show :filename`**
Example: `git show :file1.txt`
Shows the content of `file1.txt` in the staging area

> **`git show commit:filename`**
Example: `git show HEAD:file1.txt`
Shows the content of `file1.txt` in `HEAD`

`Example:` **`git show 5b80ea8:file1.txt`**
Shows the content of `file1.txt` in the commit object `5b80ea8`

Undoing...

- **`git commit --amend`** (undoing `commit`)
  Overwrites the current commit, adding the currently staged files and the current message.
- **`git reset commit <filename>`** (undoing staging)

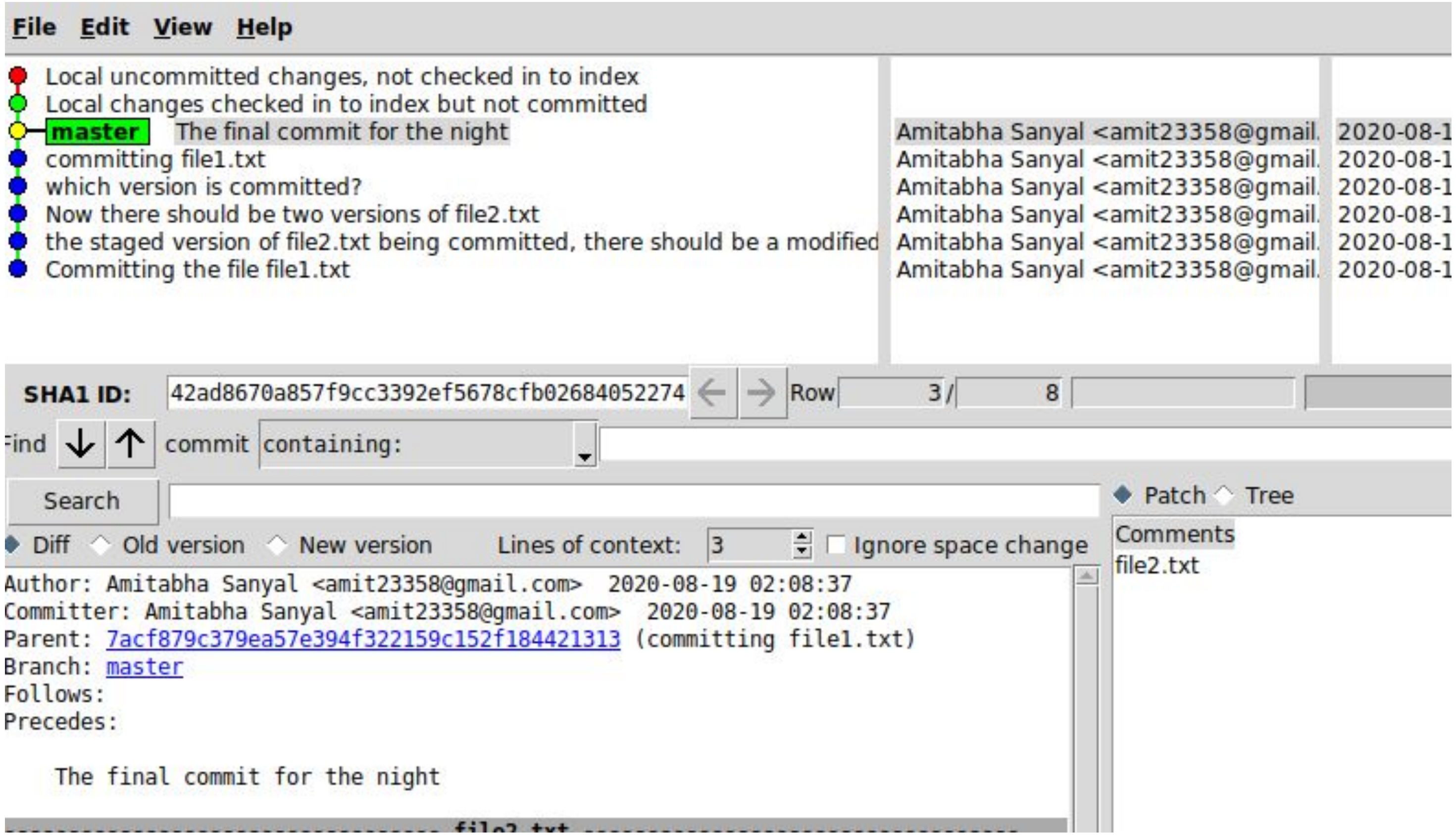
Working area	Staging area	Commit
`file.txt - v1`	`file.txt - v2`	`file.txt -v3`

- **`git checkout <filename>`** (undoing the working directory)

Working area	Staging area	Commit
`file.txt - v1`	`file.txt - v2`	`file.txt -v3`

git gui

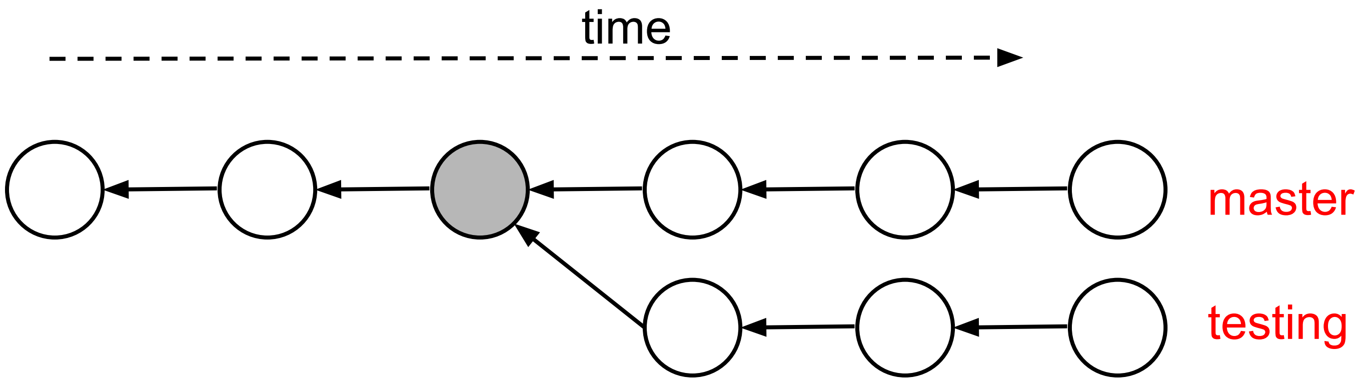
- `sudo apt-get install git-gui` (command is `gitk`)



```
Local uncommitted changes, not checked in to index
Local changes checked in to index but not committed
master  The final commit for the night                                  Amitabha Sanyal <amit23358@gmail.  2020-08-1
committing file1.txt                                                    Amitabha Sanyal <amit23358@gmail.  2020-08-1
which version is committed?                                             Amitabha Sanyal <amit23358@gmail.  2020-08-1
Now there should be two versions of file2.txt                           Amitabha Sanyal <amit23358@gmail.  2020-08-1
the staged version of file2.txt being committed, there should be a modified  Amitabha Sanyal <amit23358@gmail.  2020-08-1
Committing the file file1.txt                                           Amitabha Sanyal <amit23358@gmail.  2020-08-1

SHA1 ID: 42ad8670a857f9cc3392ef5678cfb02684052274   Row 3 / 8
Lines of context: 3

Author: Amitabha Sanyal <amit23358@gmail.com>  2020-08-19 02:08:37
Committer: Amitabha Sanyal <amit23358@gmail.com>  2020-08-19 02:08:37
Parent: 7acf879c379ea57e394f322159c152f184421313 (committing file1.txt)
Branch: master
Follows:
Precedes:

    The final commit for the night

---------------------------------- file2.txt ----------------------------------
```

Branching

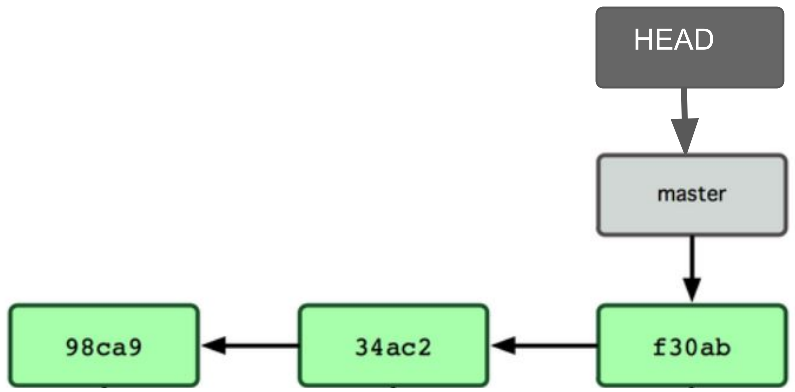
- A branch is a sequential line of development
- Introducing a new branch means starting a new line of development that does not interfere with the original line.
    - The new branch may be merged with the original
    - Enables parallel development of new ideas

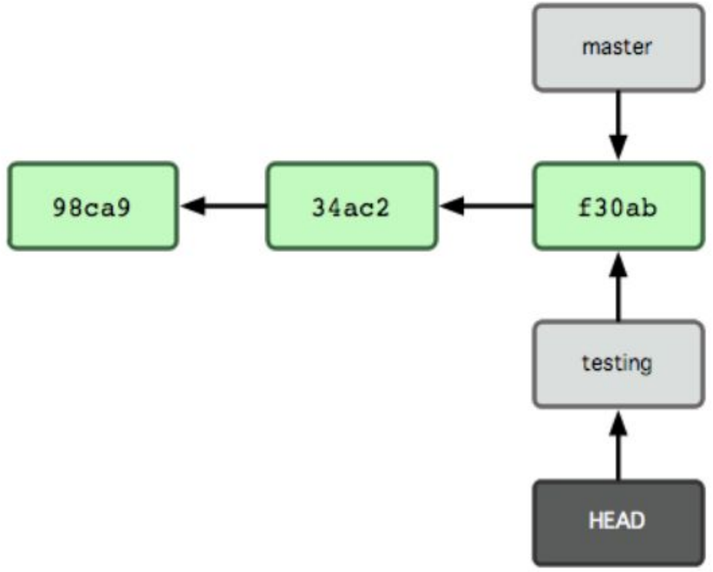


Structure of a branch

- HEAD points to the current branch

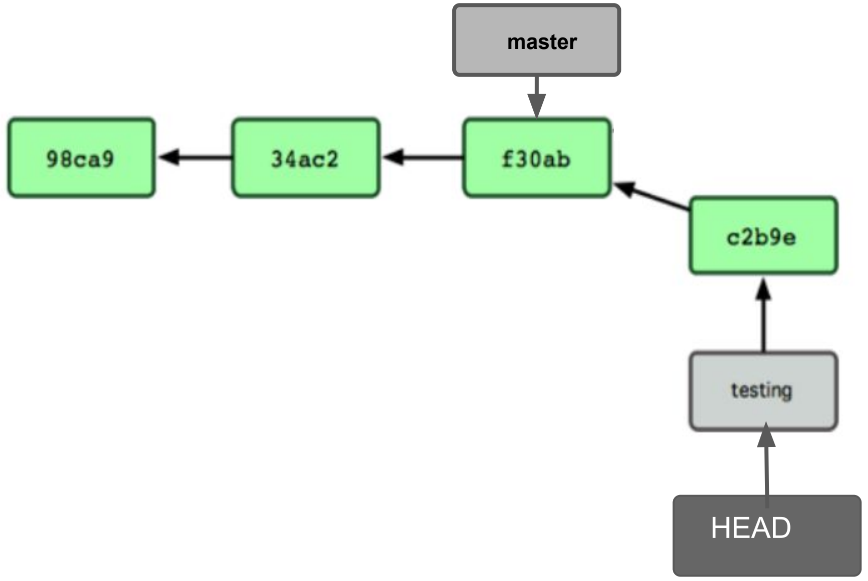
```
> git checkout -b testing
```



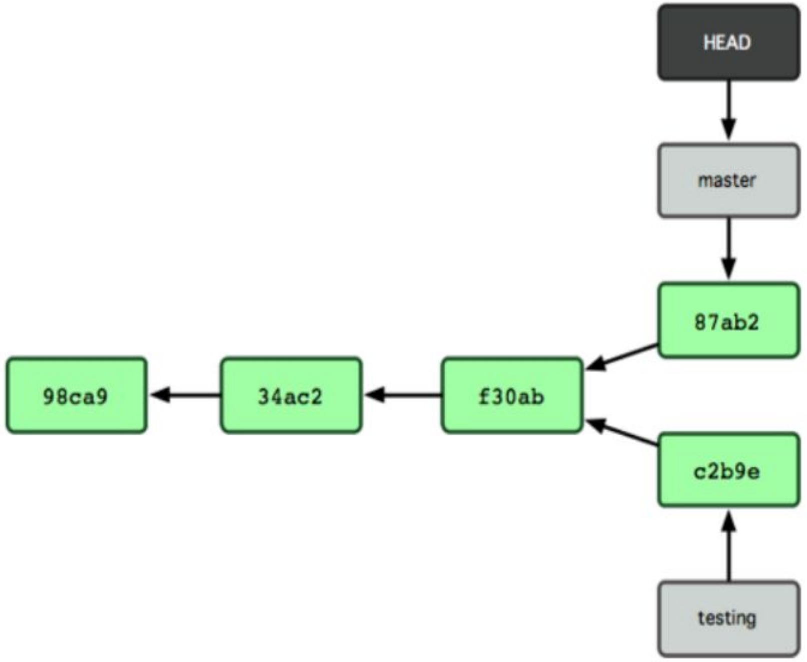


Separate developments

Development along testing



Separate development along master



Useful branch commands

- **`git checkout -b bname`**
  `Introduces a new branch bname which becomes the current branch`

- **`git checkout bname`**
  `makes an existing branch bname the current branch`

- **`git branch`**
  `lists all branches`

- **`git branch -d bname`**
  `deletes the branch bname`

`git stash`

If we want to change branch without committing, to preserve the current status, we have to `stash`.

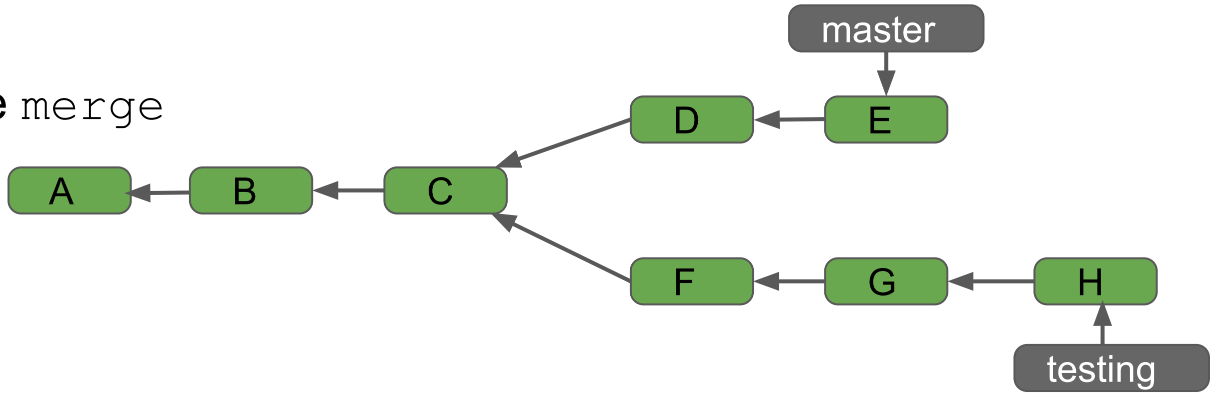
```
> git stash
```
stashes or stores the current working area and staging area on a stack

```
> git stash list
```
show the stack of stashes

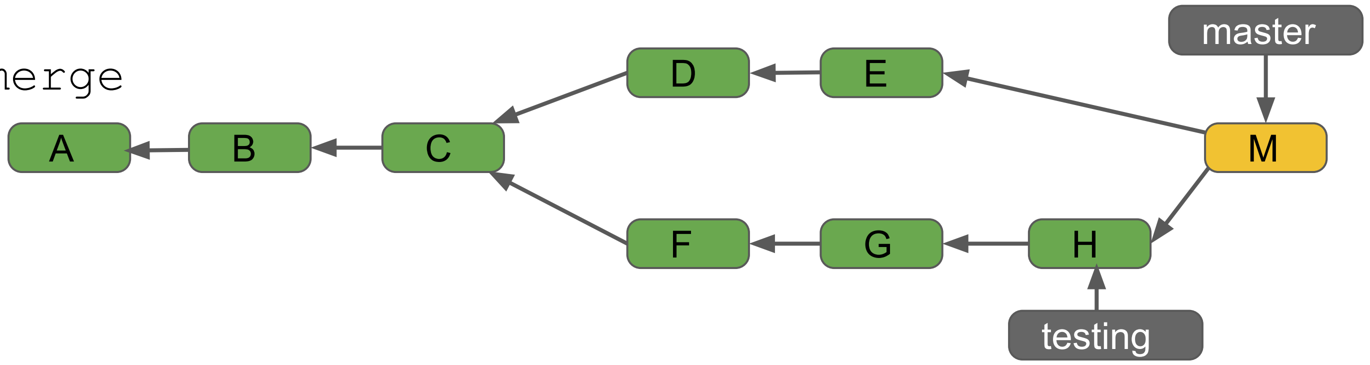
```
> git stash apply stash@{n} --index
```
restores **`n`**th stash from the stack

`git merge`

- Before `merge`

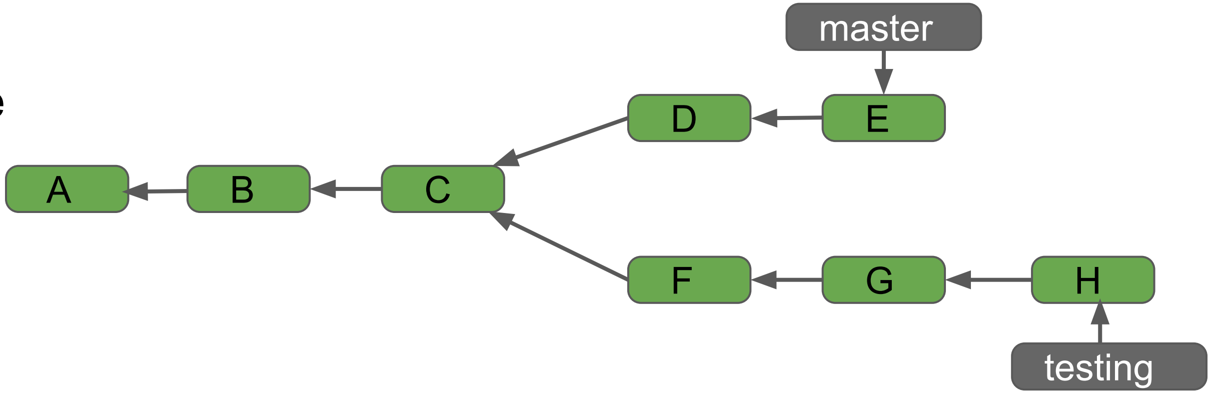


- After `merge`

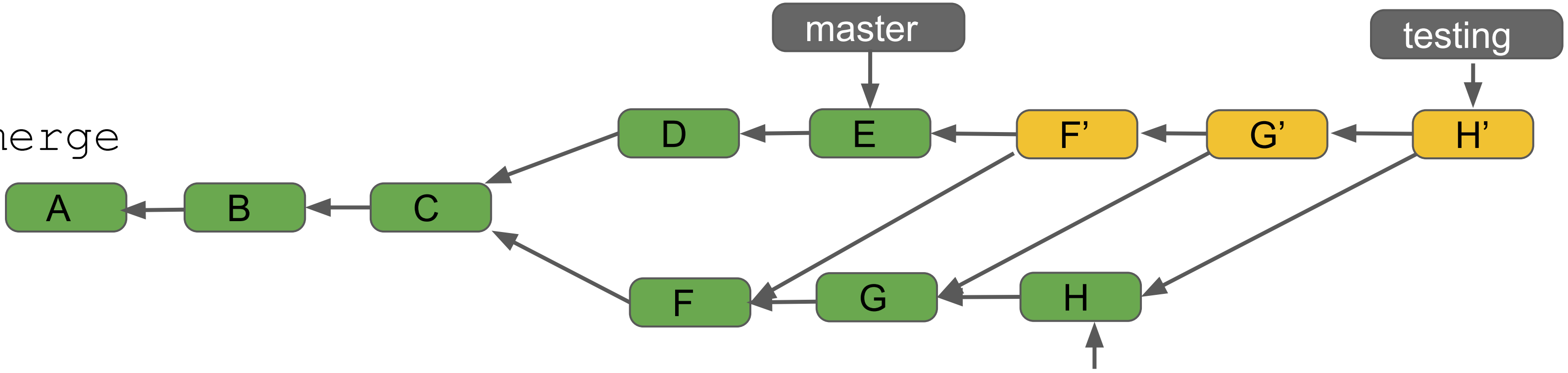


git rebase

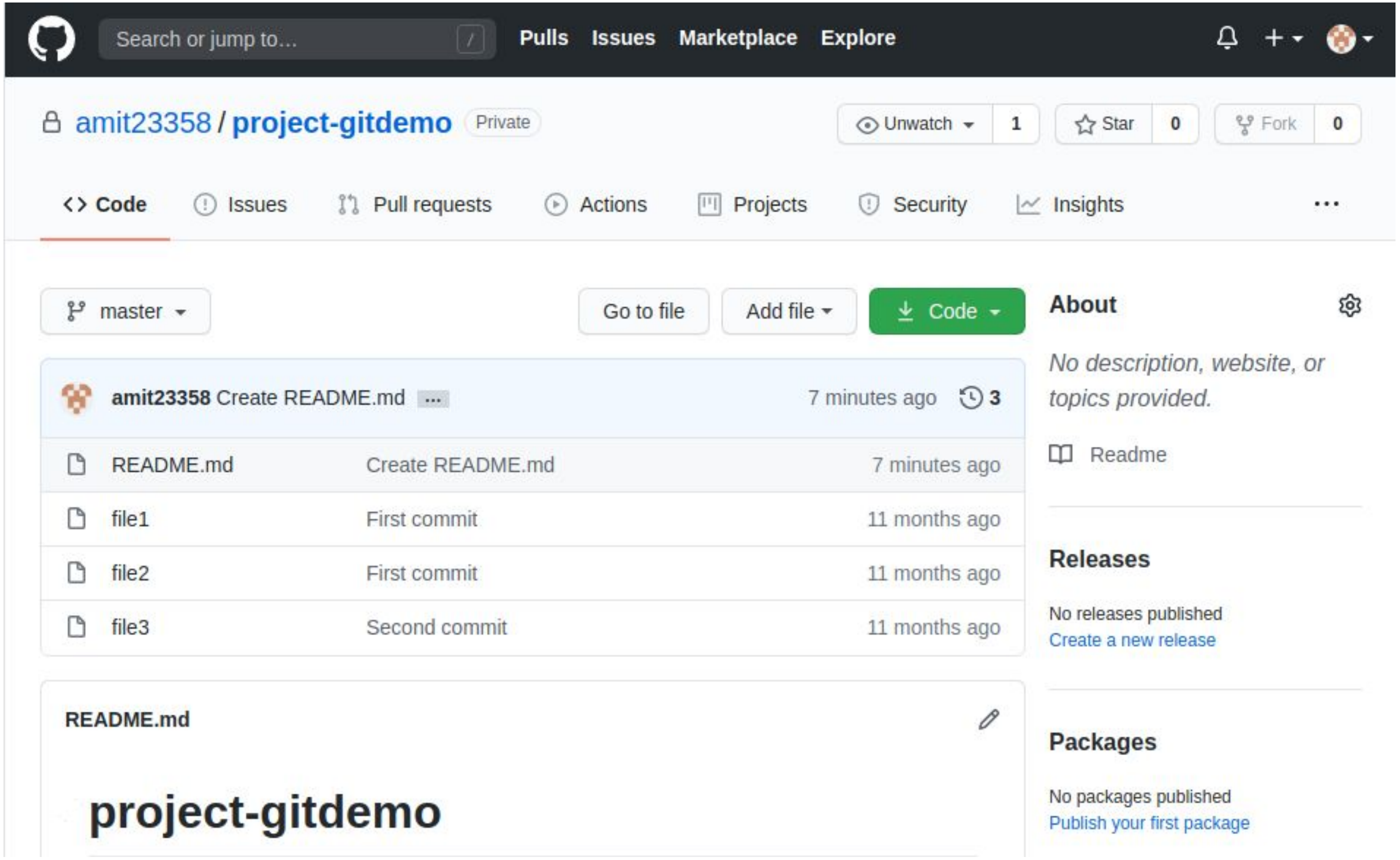
- Before



- After `merge`



Working with remote repositories



amit23358 / project-gitdemo Private

amit23358 Create README.md ... 7 minutes ago 3

README.md	Create README.md	7 minutes ago
file1	First commit	11 months ago
file2	First commit	11 months ago
file3	Second commit	11 months ago

README.md

project-gitdemo

About

No description, website, or topics provided.

Readme

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Working with remote repositories

- `git clone https://github.com/amit23358/project-gitdemo.git`
  - Clones an existing project called `project-gitdemo.git`

Back to our freshly created repo demo-CS251-2020

- `git remote add origin` https://github.com/amit23358/demo-cS251-2020.git
  - Links the local repo with the remote repo. Also gives it a name “origin”
- `git fetch origin`
  - Brings in meta-data but not files
- `git push origin master`
  - Pushes the local repo to the remote repo
- `git pull origin master`
  - Pulls in files from remote repo to local repo

Outline