git

About Git

Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

Version control systems

- Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all "real" projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a "central" repository; users "check out" files, work on them, and "check them in"
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

Why version control?

- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes
- For getting an internship or job:
  - Any company with a clue uses some kind of version control
  - Companies without a clue are bad places to work

Why Git?

- Git has many advantages over earlier systems such as CVS and Subversion
  - More efficient, better workflow, etc.
  - Of course, there are always those who disagree
- Best competitor: Mercurial
  - Same concepts, slightly simpler to use
  - Much less popular than Git

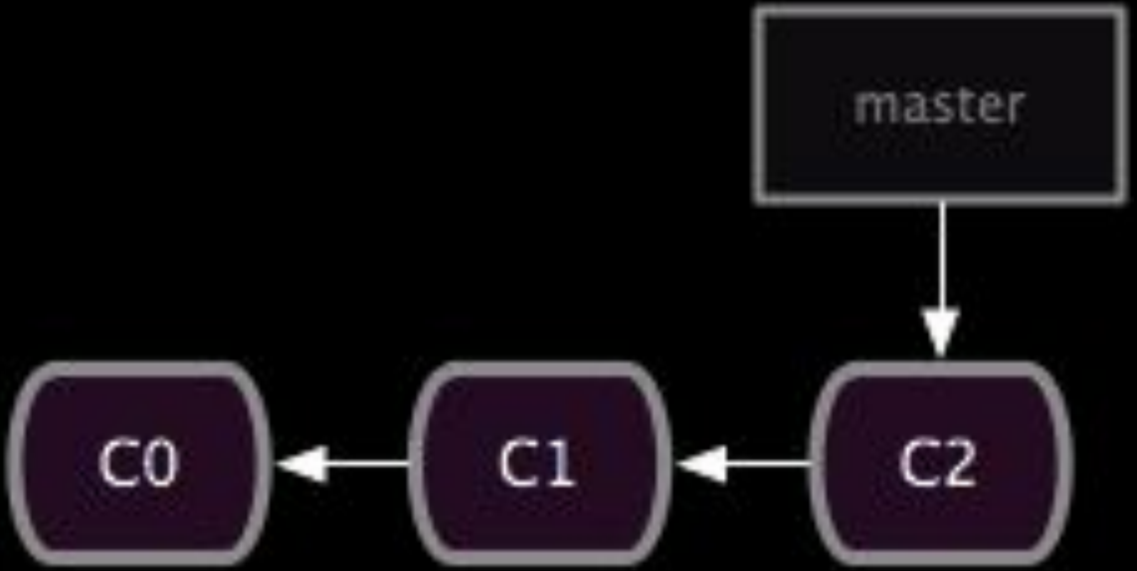
Download and install Git

- There are online materials that are better than any that I could provide
- Here's the standard one: http://git-scm.com/downloads
- Here's one from StackExchange: http://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide#323764

- Note: Git is primarily a command-line tool

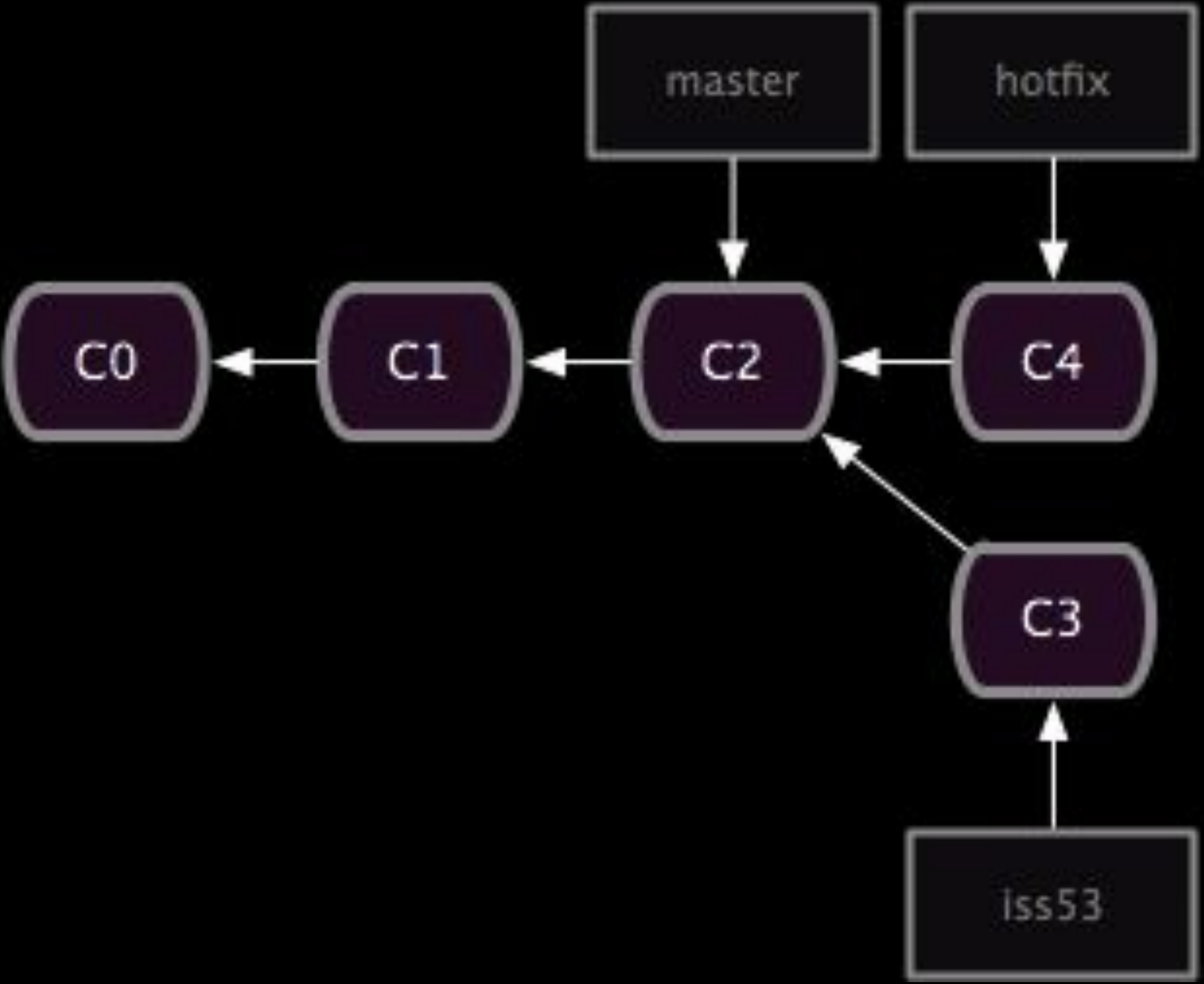
Why track/manage revisions?

Backup: Undo or refer to old stuff



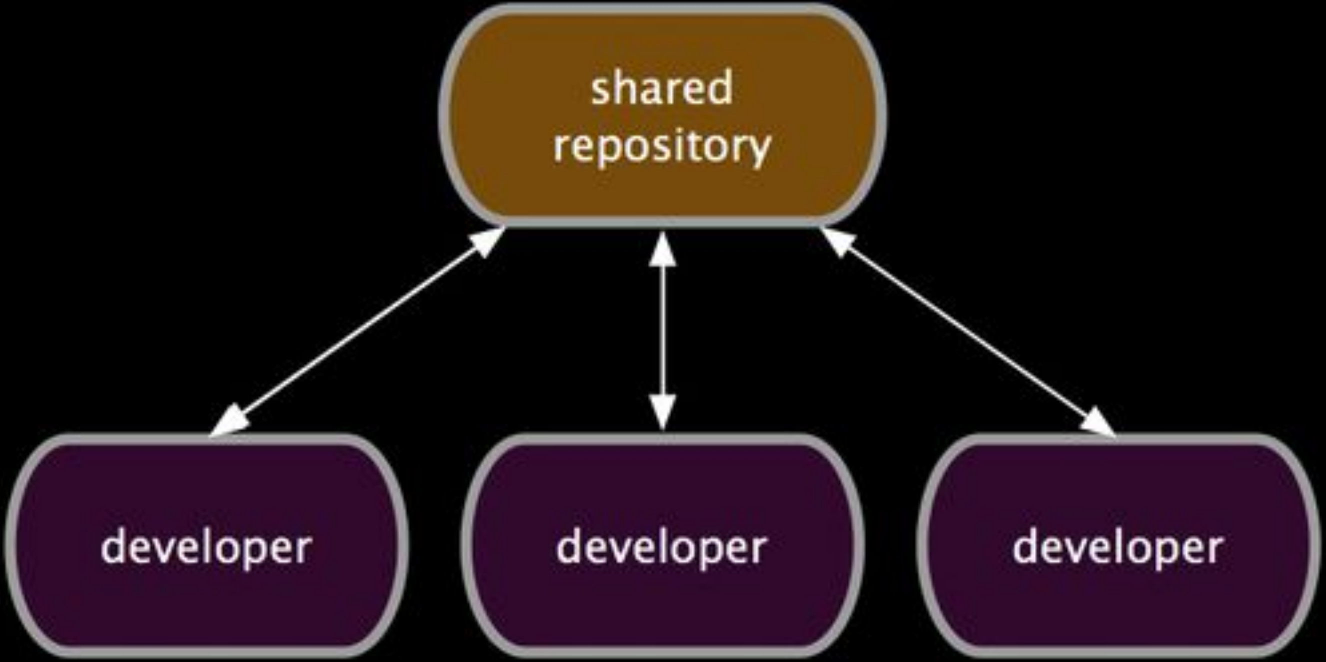
http://git-scm.com/book/en/Git-Branching-Basic-Branching-and-Merging

Branch: Maintain old release while working on new



http://git-scm.com/book/en/Git-Branching-Basic-Branching-and-Merging

Collaborate: Work in parallel with teammates



http://git-scm.com/book/en/Distributed-Git-Distributed-Workflows

Version Control Systems (VCSs)

- Help you track/manage/distribute revisions
- Standard in modern development
- Examples:
  - Revision Control System (RCS)
  - Concurrent Versions System (CVS)
  - Subversion (SVN)
  - Git

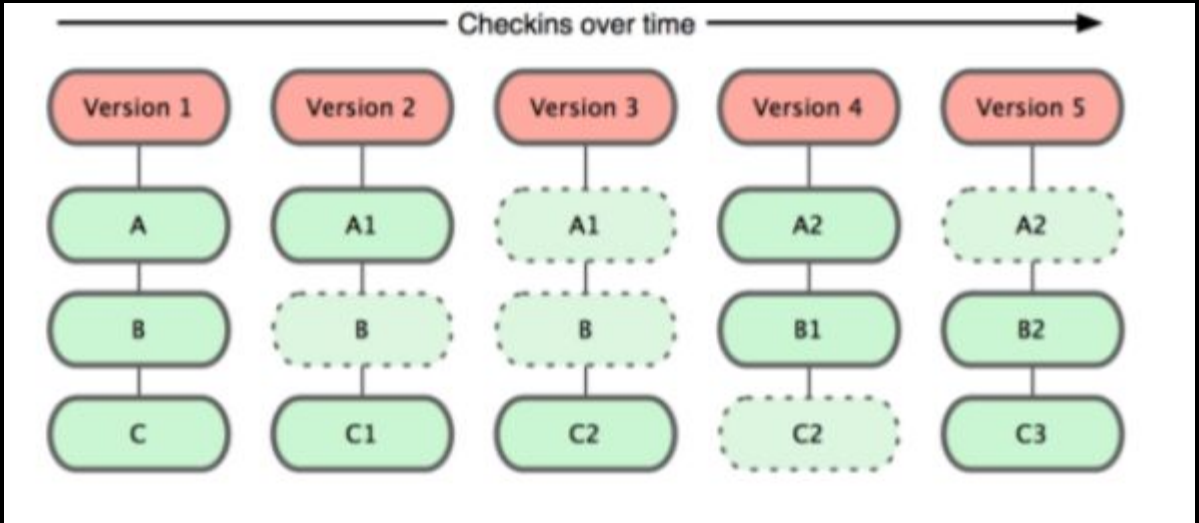
older

newer

Our focus

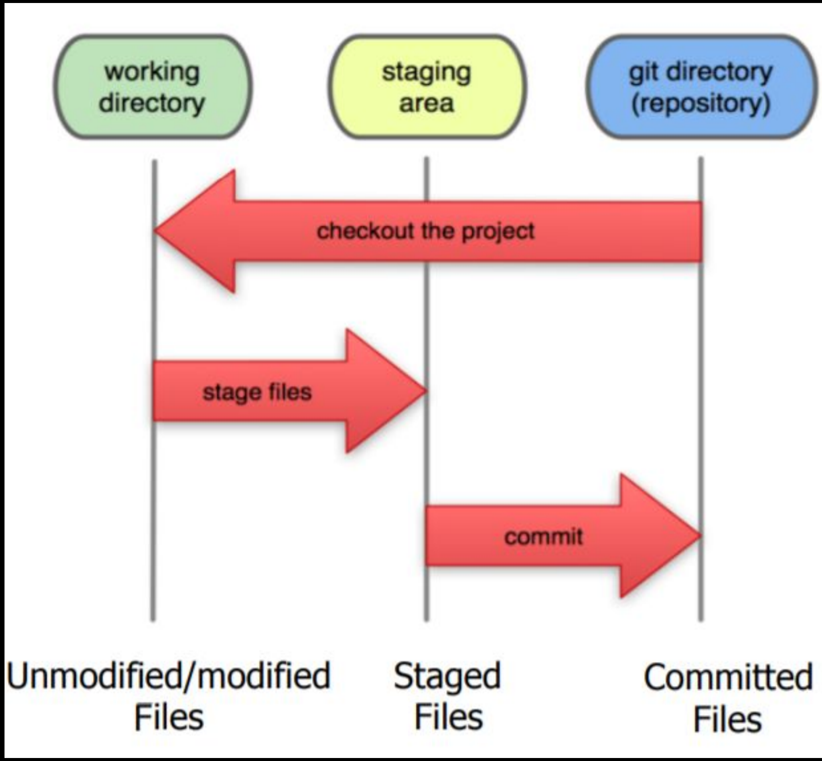
Git snapshots

• Git keeps "snapshots" of the entire state of the project.
– Each checkin version of the overall code has a copy of each file in it.
– Some files change on a given checkin, some do not.
– More redundancy, but faster.



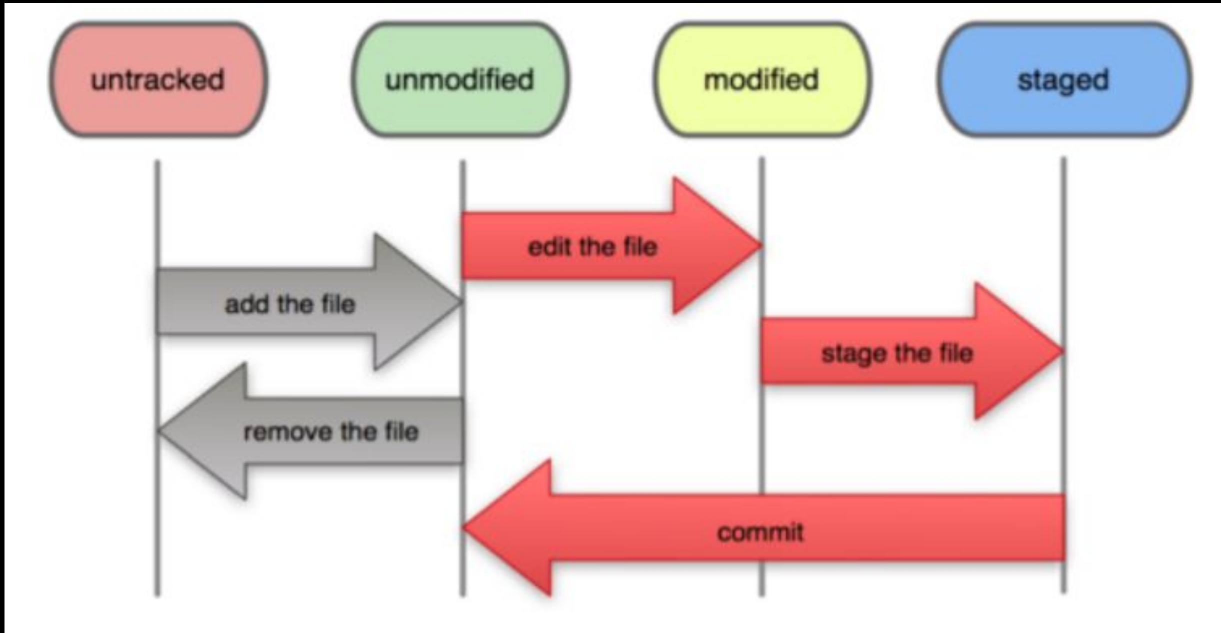
Local git areas

In your local copy on git,
files can be:
– In your local repo
  - (committed)
– Checked out and modified,
but not yet committed
  - (working copy)
– Or, in-between, in
a "staging" area
  - Staged files are ready
to be committed.
  - A commit saves a snapshot
of all staged state.



Basic Git workflow

- **Modify** files in your working directory.
- **Stage** files, adding snapshots of them to your staging area.
- **Commit**, which takes the files in the staging area and stores that snapshot permanently to your Git directory.



Initial Git configuration

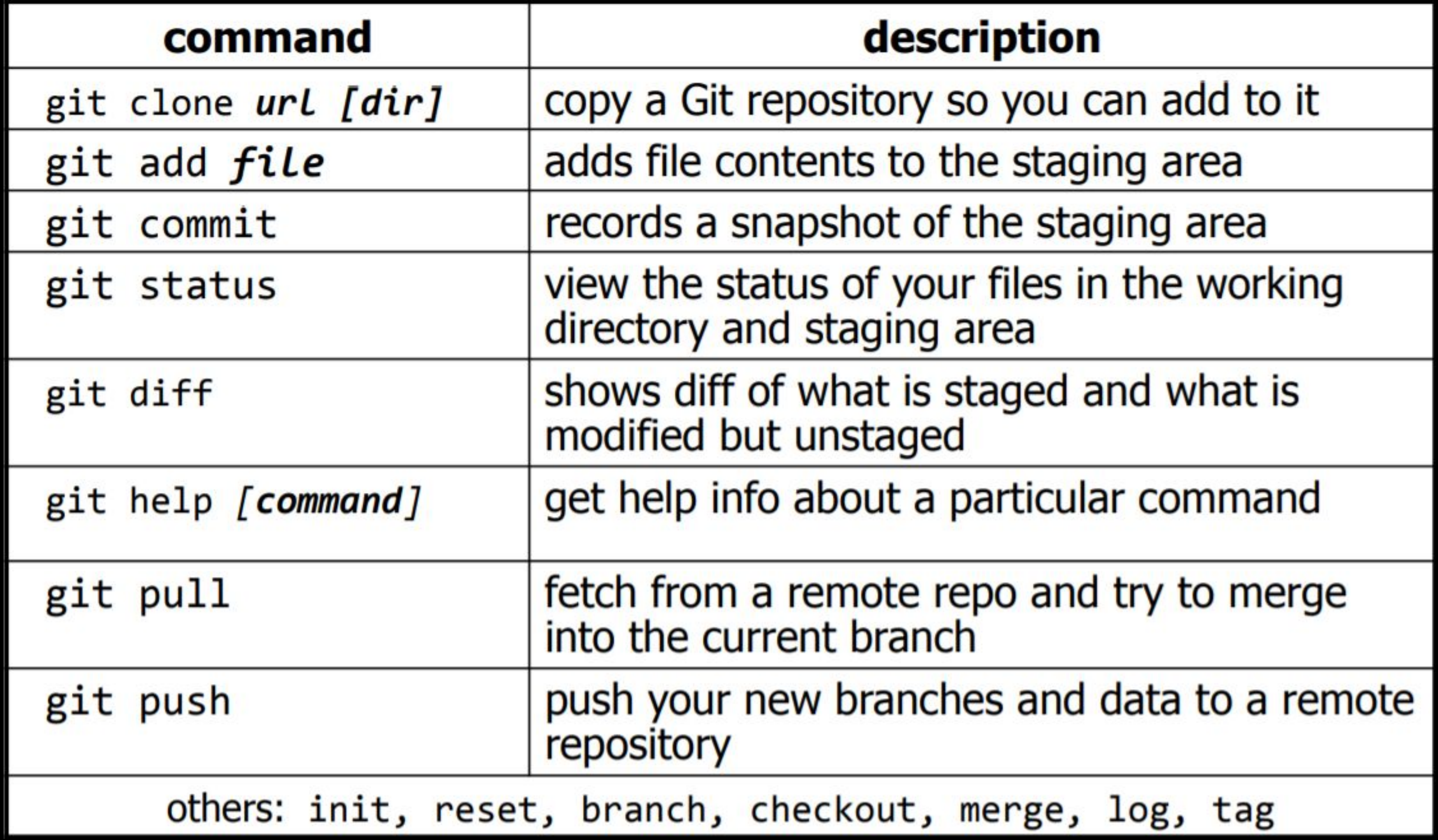
- Set the name and email for Git to use when you commit:
- - `git config --global user.name "Bugs Bunny"`
- – `git config --global user.email bugs@gmail.com`
- – You can call `git config –list` to verify these are set.

Creating a Git repo

Two common scenarios: (only do one of these)

- To create a new local Git repo in your current directory:
  - `git init`
  - This will create a .git directory in your current directory.
- Then you can commit files in that directory into the repo.
  - `git add filename`
  - `git commit -m "commit message"`
- To clone a remote repo to your current directory:
  - `git clone url localDirectoryName`
  - This will create the given local directory, containing a working copy of the files from the repo, and a .git directory (used to hold the staging area and your actual local repo)

Git commands



command	description
`git clone` ***`url [dir]`***	copy a Git repository so you can add to it
`git add` ***`file`***	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help` ***`[command]`***	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init`, `reset`, `branch`, `checkout`, `merge`, `log`, `tag`	

Add and commit a file

- The first time we ask a file to be tracked, and every time before we commit a file, we must add it to the staging area:
  - `git add Hello.java Goodbye.java`
  - Takes a snapshot of these files, adds them to the staging area.
  - In older VCS, "add" means "start tracking this file."
  - In Git, "add" means "add to staging area" so it will be part of the next commit.
- To move staged changes into the repo, we commit:
  - `git commit -m "Fixing bug #22"`
- To undo changes on a file before you have committed it:
  - `git reset HEAD -- filename (unstages the file)`
  - `git checkout -- filename (undoes your changes)`
- All these commands are acting on your local version of repo.

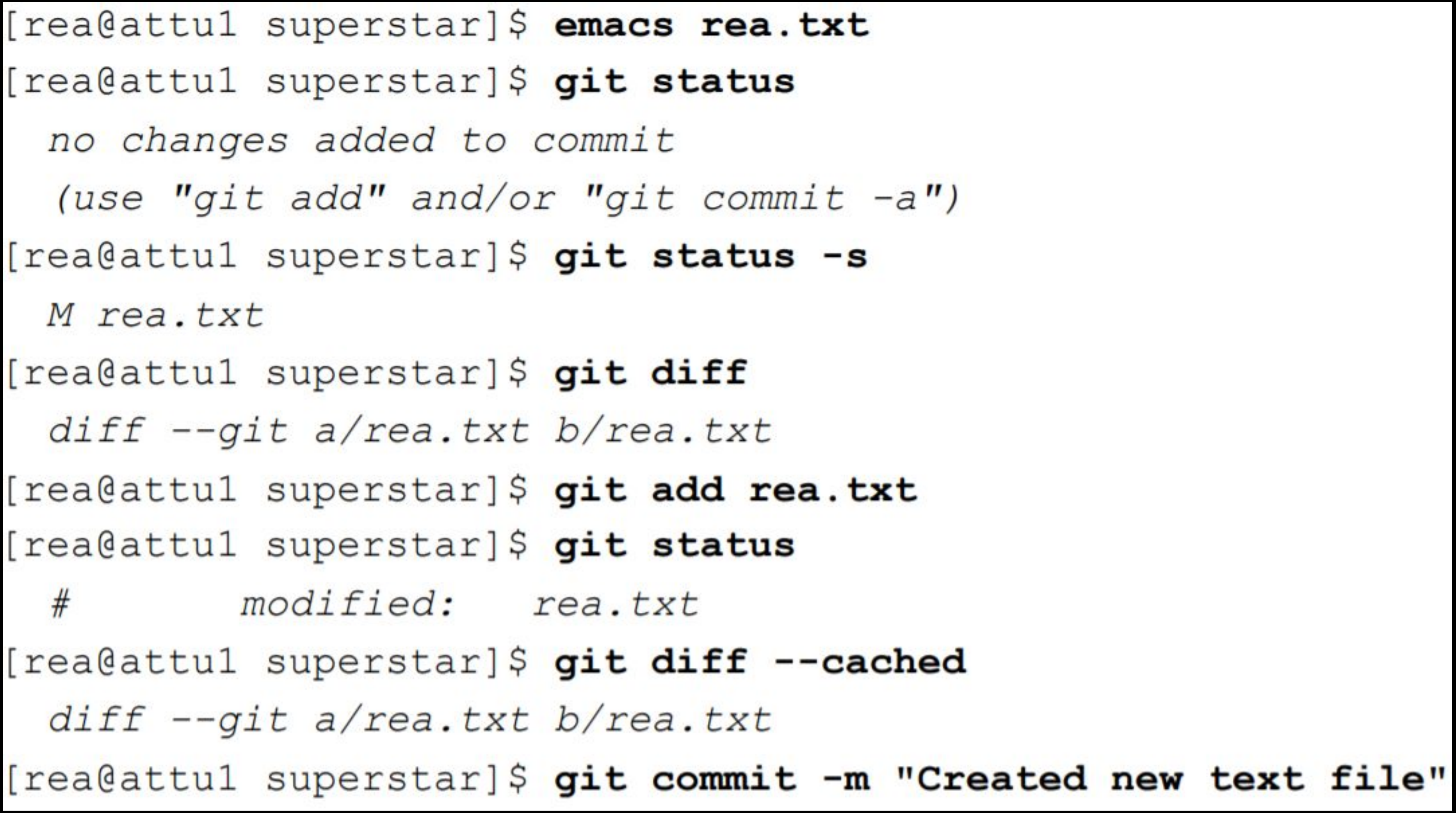
Viewing/undoing changes

- To view status of files in working directory and staging area:
  - `git status` or `git status -s` (short version)
- To see what is modified but unstaged:
  - `git diff`
- To see a list of staged changes:
  - `git diff --cached`
- To see a log of all changes in your local repo:
  - `git log` or `git log --oneline` (shorter version)

```
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

- `git log -5` (to show only the 5 most recent updates), etc.

An example workflow



```
[rea@attu1 superstar]$ emacs rea.txt
[rea@attu1 superstar]$ git status
  no changes added to commit
  (use "git add" and/or "git commit -a")
[rea@attu1 superstar]$ git status -s
  M rea.txt
[rea@attu1 superstar]$ git diff
  diff --git a/rea.txt b/rea.txt
[rea@attu1 superstar]$ git add rea.txt
[rea@attu1 superstar]$ git status
  #       modified:   rea.txt
[rea@attu1 superstar]$ git diff --cached
  diff --git a/rea.txt b/rea.txt
[rea@attu1 superstar]$ git commit -m "Created new text file"
```

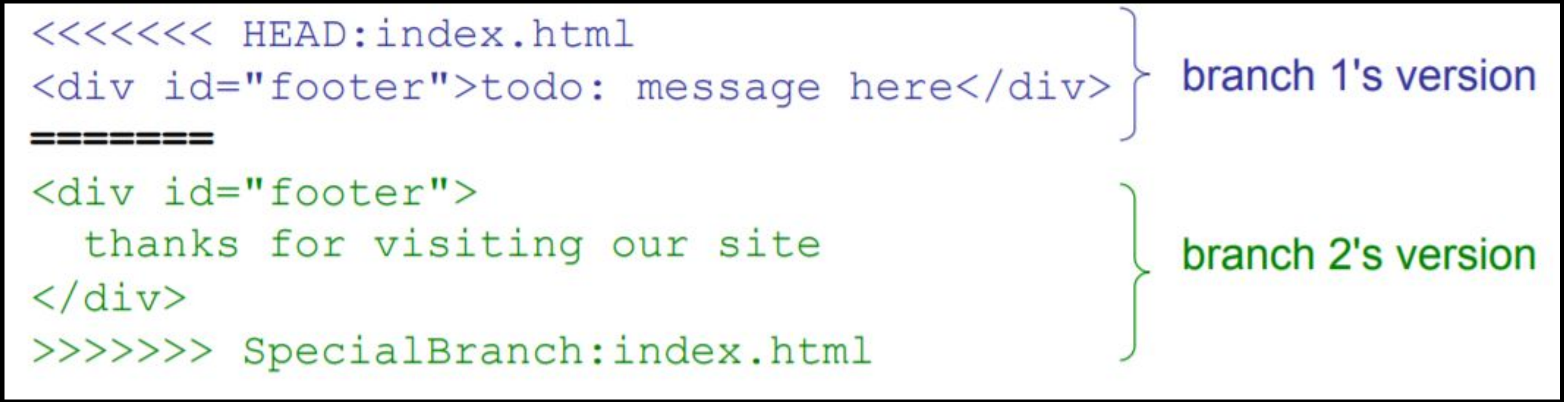
Branching and merging

Git uses branching heavily to switch between multiple tasks.

- To create a new local branch:
  - `git branch name`
- To list all local branches: (* = current branch)
  - `git branch`
- To switch to a given local branch:
  - `git checkout branchname`
- To merge changes from a branch into the local master:
  - `git checkout master`
  - `git merge branchname`

Merge conflicts

- The conflicting file will contain <<< and >>> sections to indicate where Git was unable to resolve a conflict:



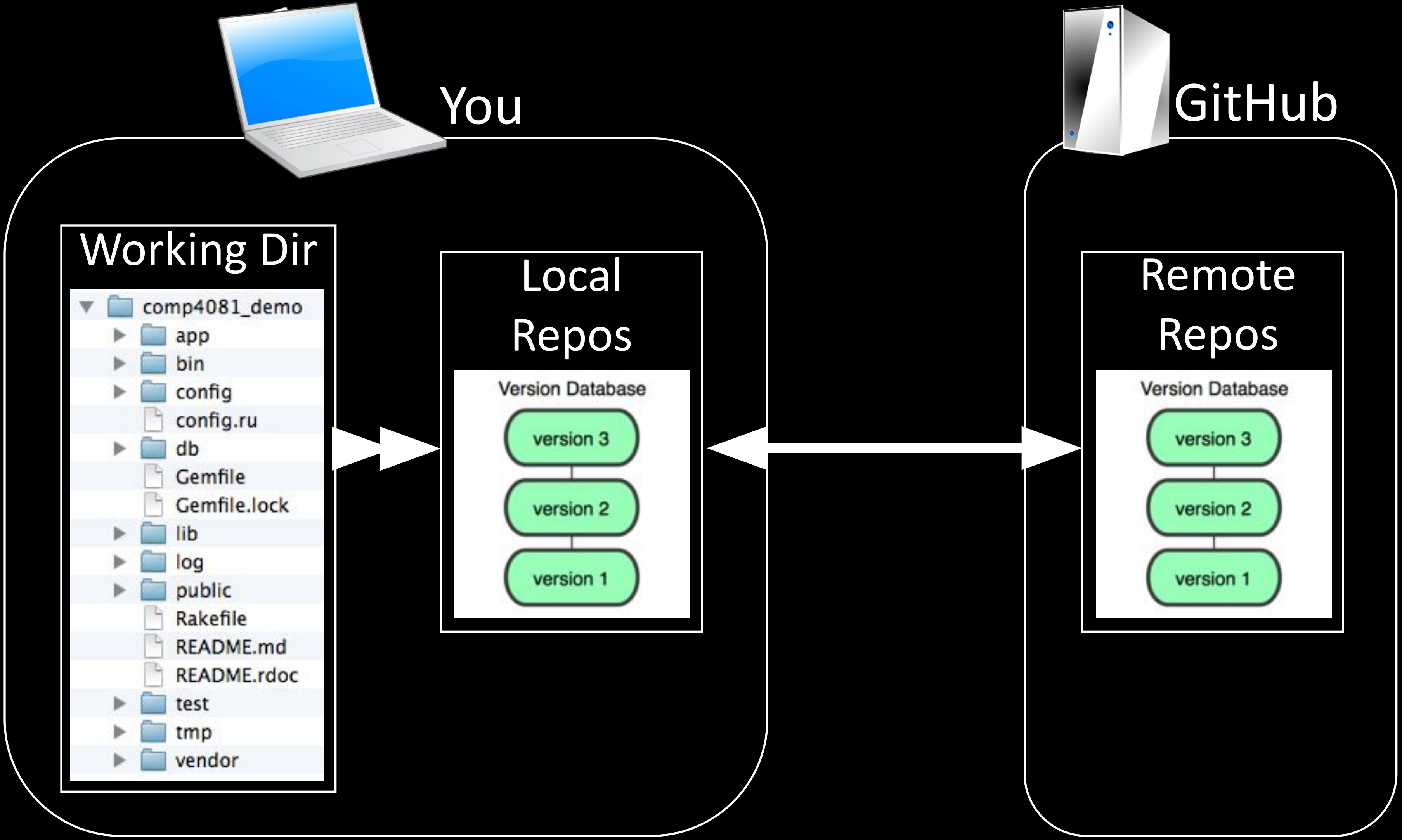
```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
  thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

- Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct).

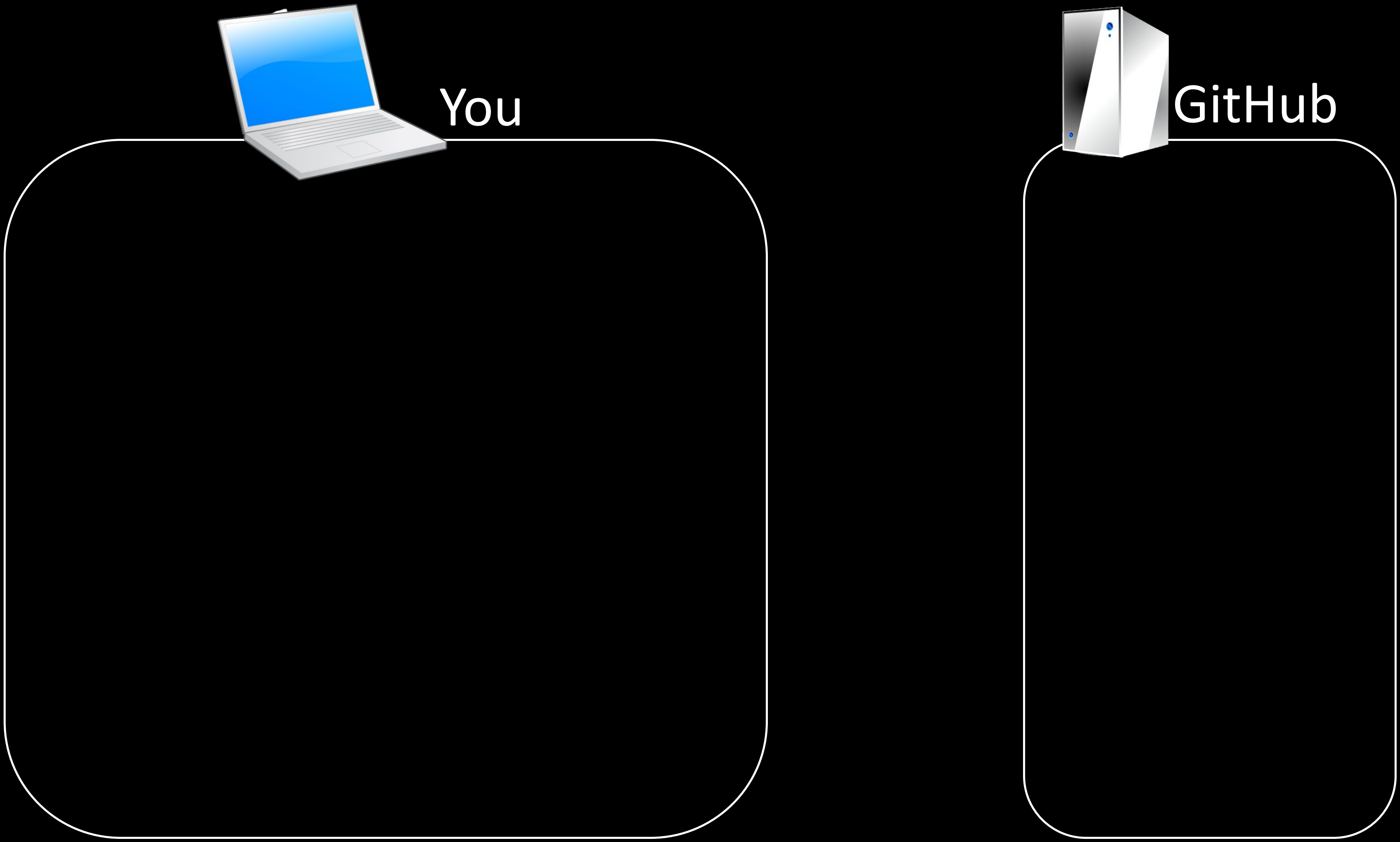
Interaction with remote repo

- **Push** your local changes to the remote repo.
- **Pull** from remote repo to get most recent changes.
  - (fix conflicts if necessary, add/commit them to your local repo)
- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - `git pull origin master`
- To put your changes from your local repo in the remote repo:
  - `git push origin master`

GitHub-User Perspective



Let’s begin with an example…



Configure your Git client
(Rails Tutorial 1.3.1)

- Install Git
- Check config info:

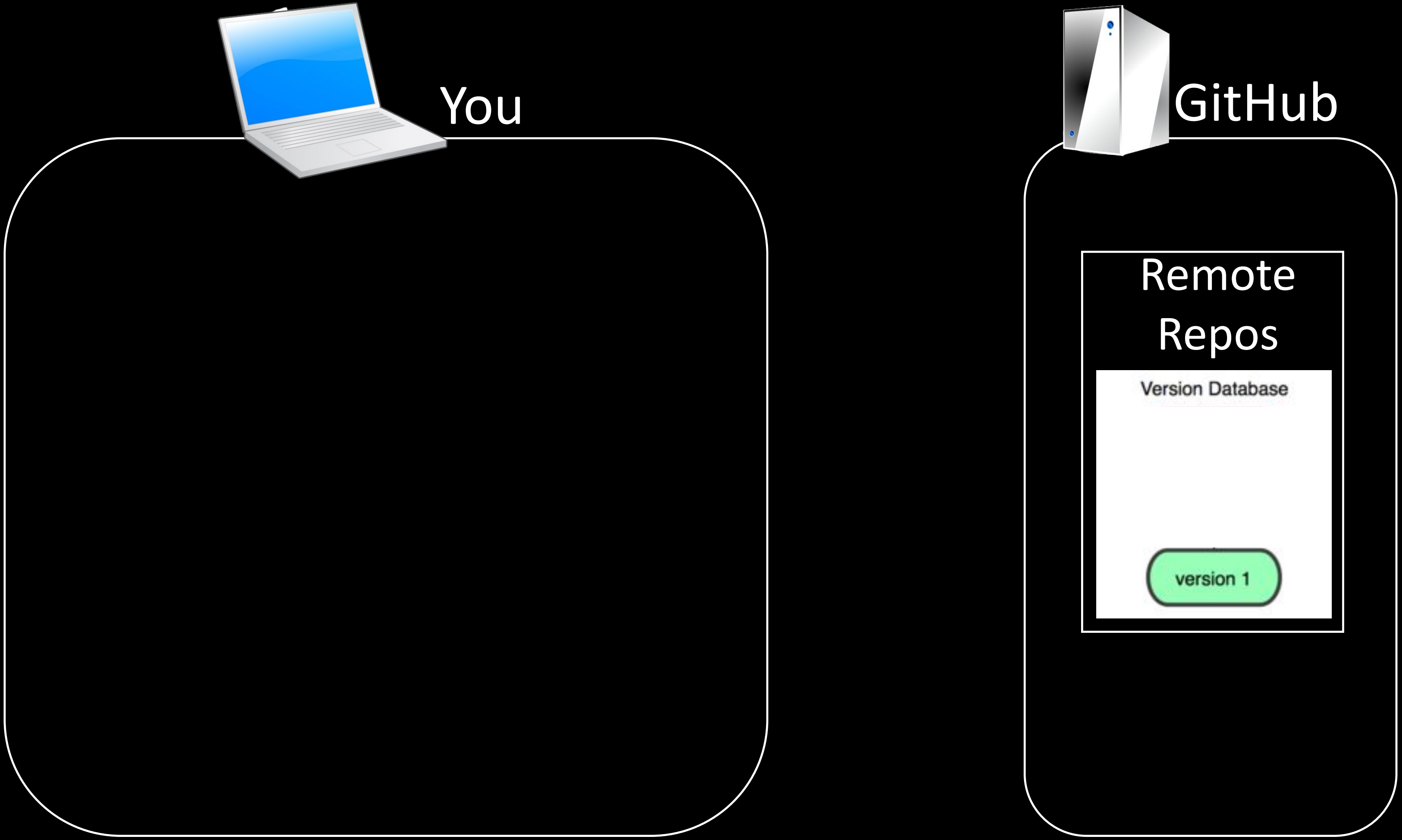
```
$ git config --list
user.name=Scott Fleming
user.email=Scott.Fleming@memphis.edu
```

- Fix if necessary:

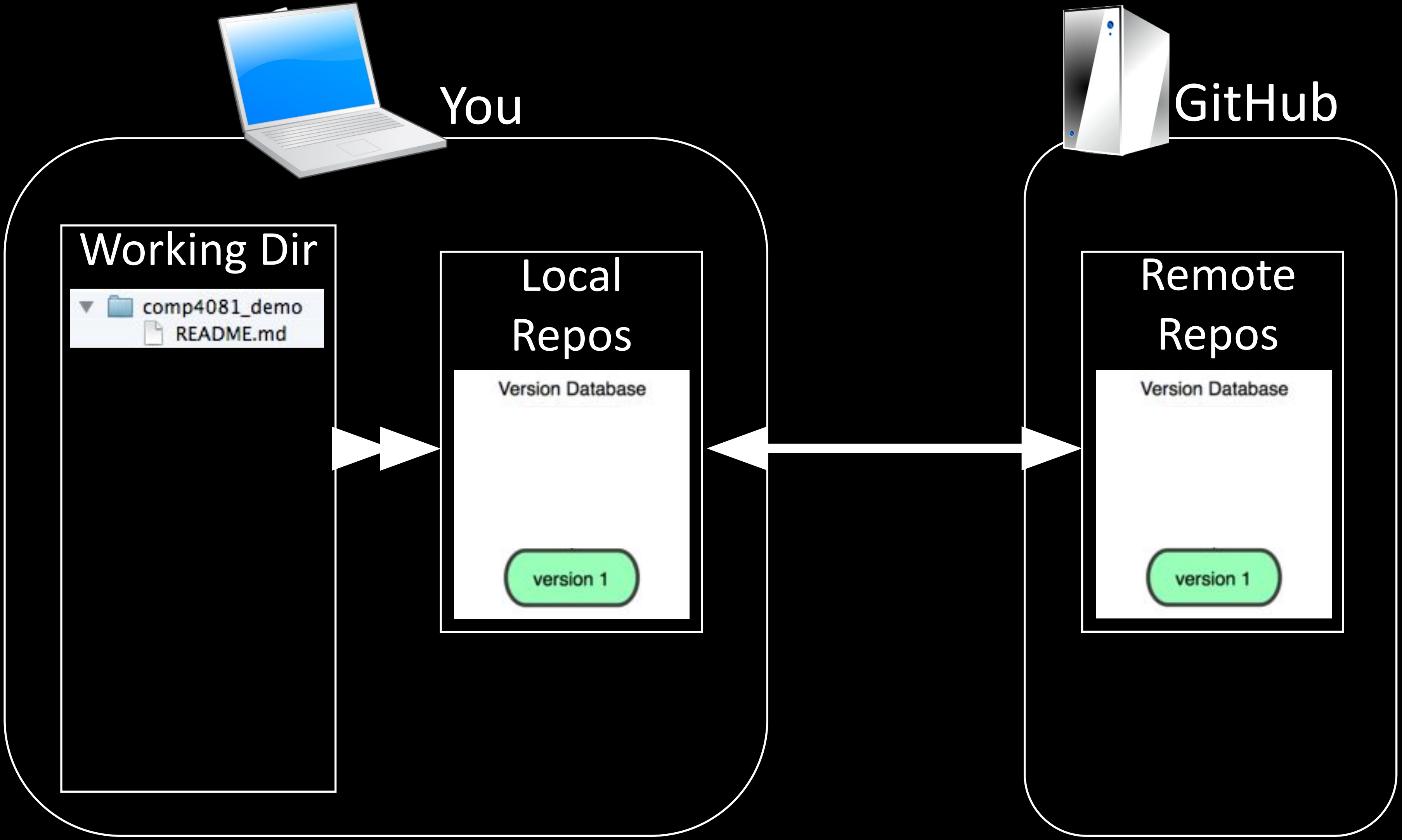
```
$ git config --global user.name "John Doe"

$ git config --global user.email jdoe@memphis.edu
```

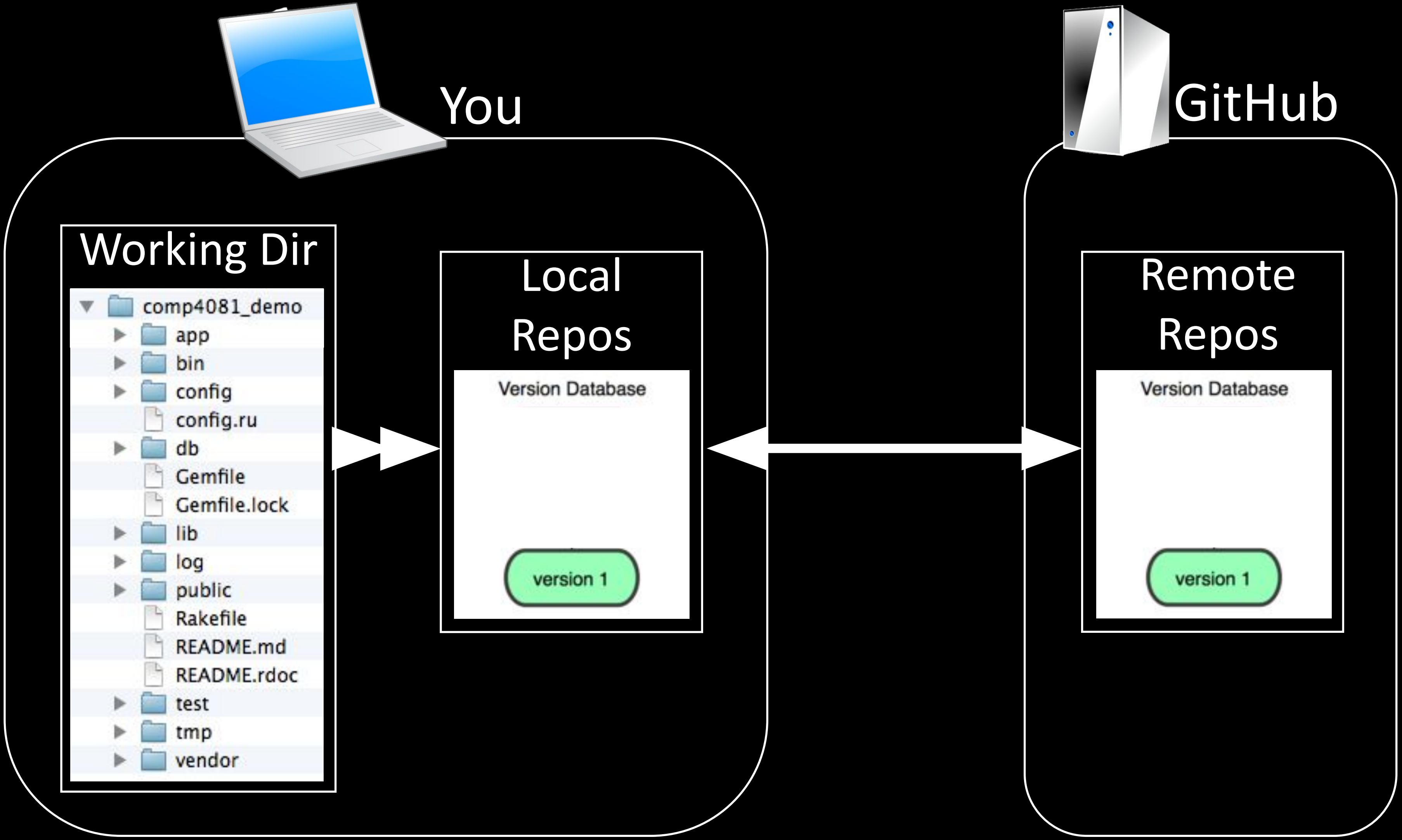
Log into GitHub and create a repos
(with add README option)



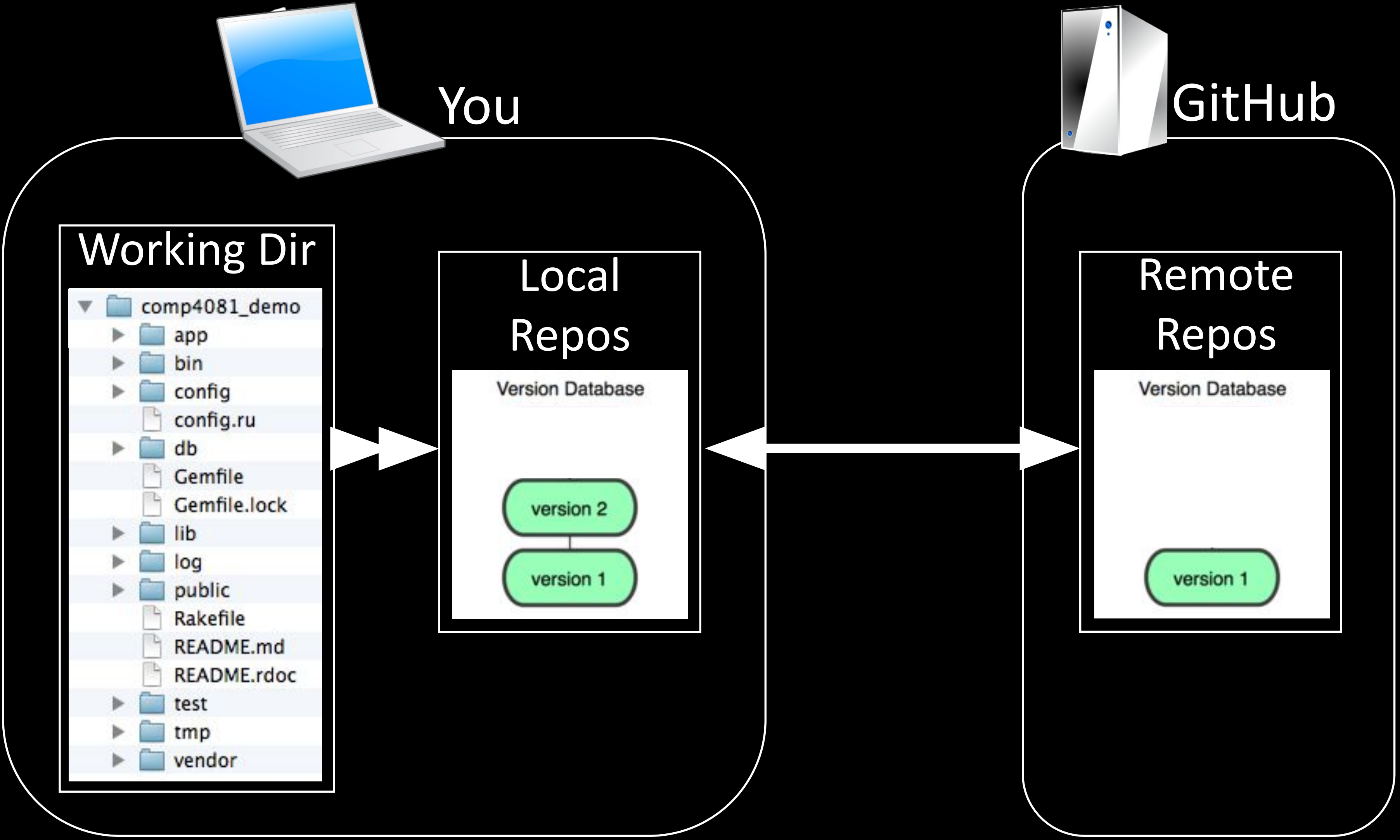
```
$ git clone https://github.com/sdflem/comp4081_demo.git
```



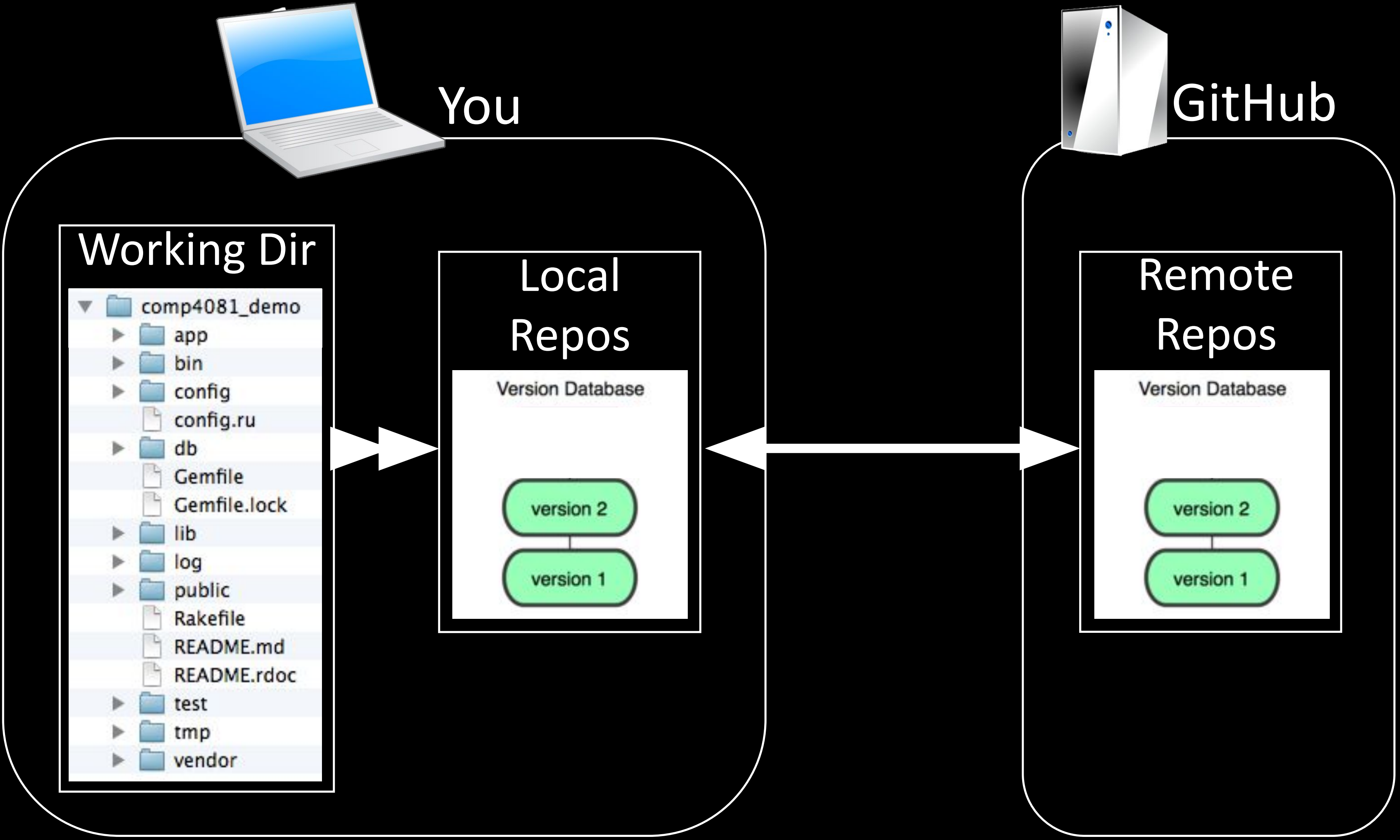
```
$ rails new comp4081_demo
```



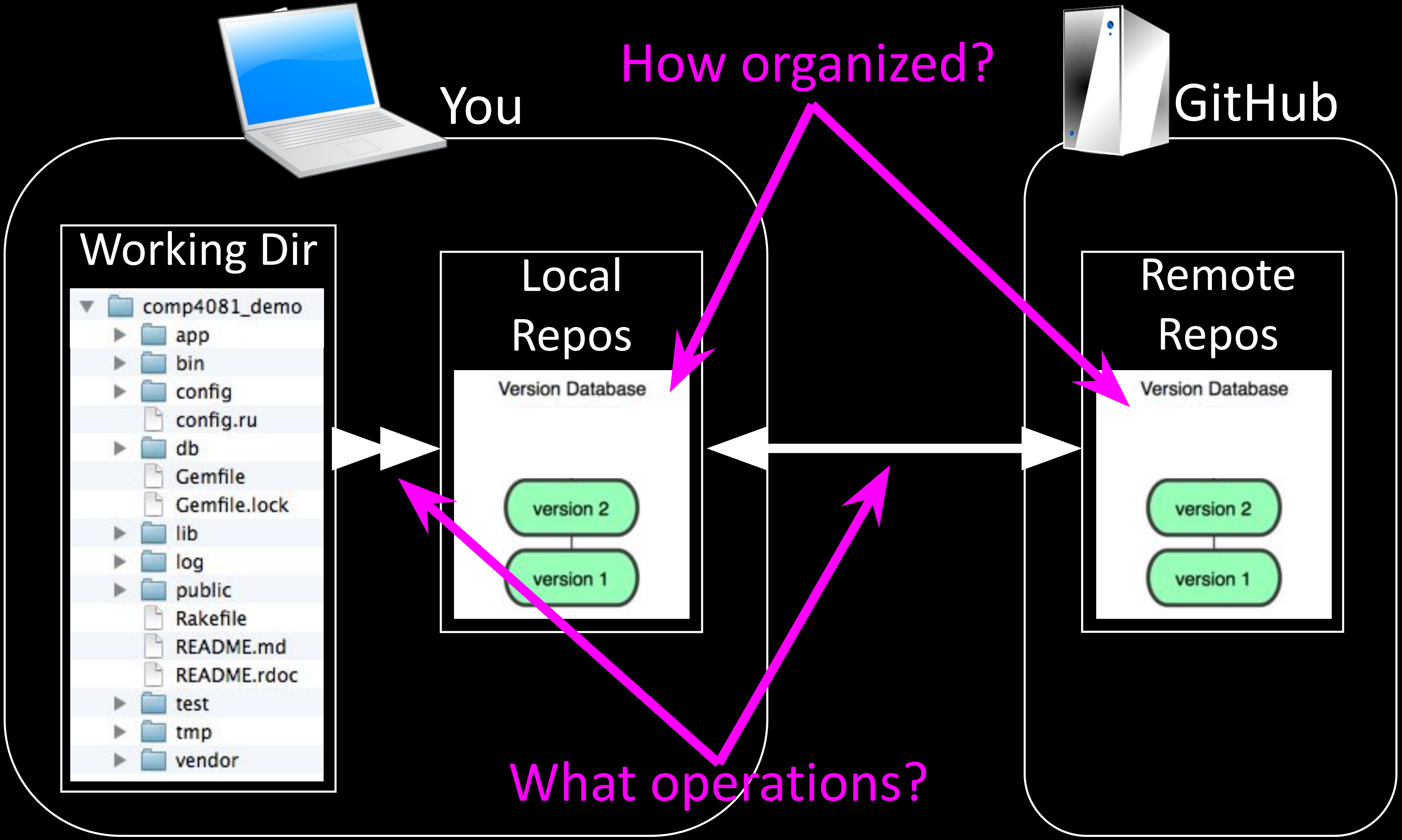
```
$ cd comp4081_demo
$ git add -A
$ git commit -m "Created Rails project skeleton"
```



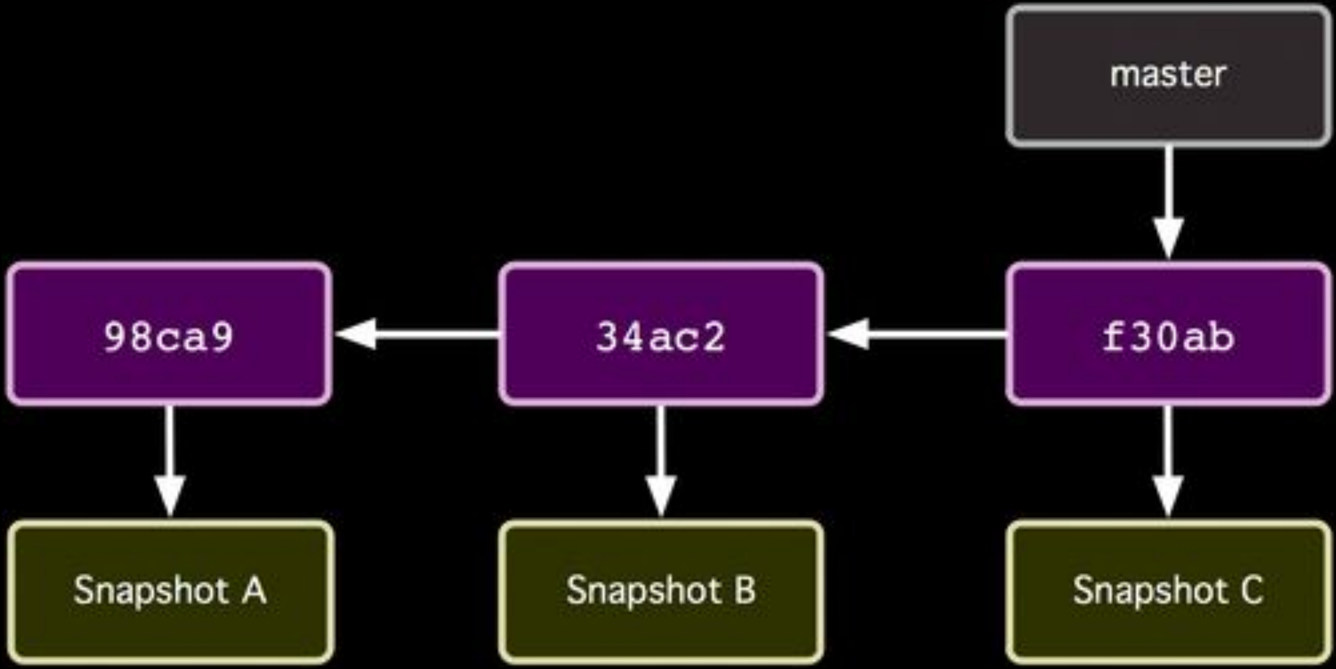
```
$ git push
```


You
GitHub
Working Dir
comp4081_demo
app
bin
config
config.ru
db
Gemfile
Gemfile.lock
lib
log
public
Rakefile
README.md
README.rdoc
test
tmp
vendor
Local Repos
Version Database
version 2
version 1
Remote Repos
Version Database
version 2
version 1

Questions to answer



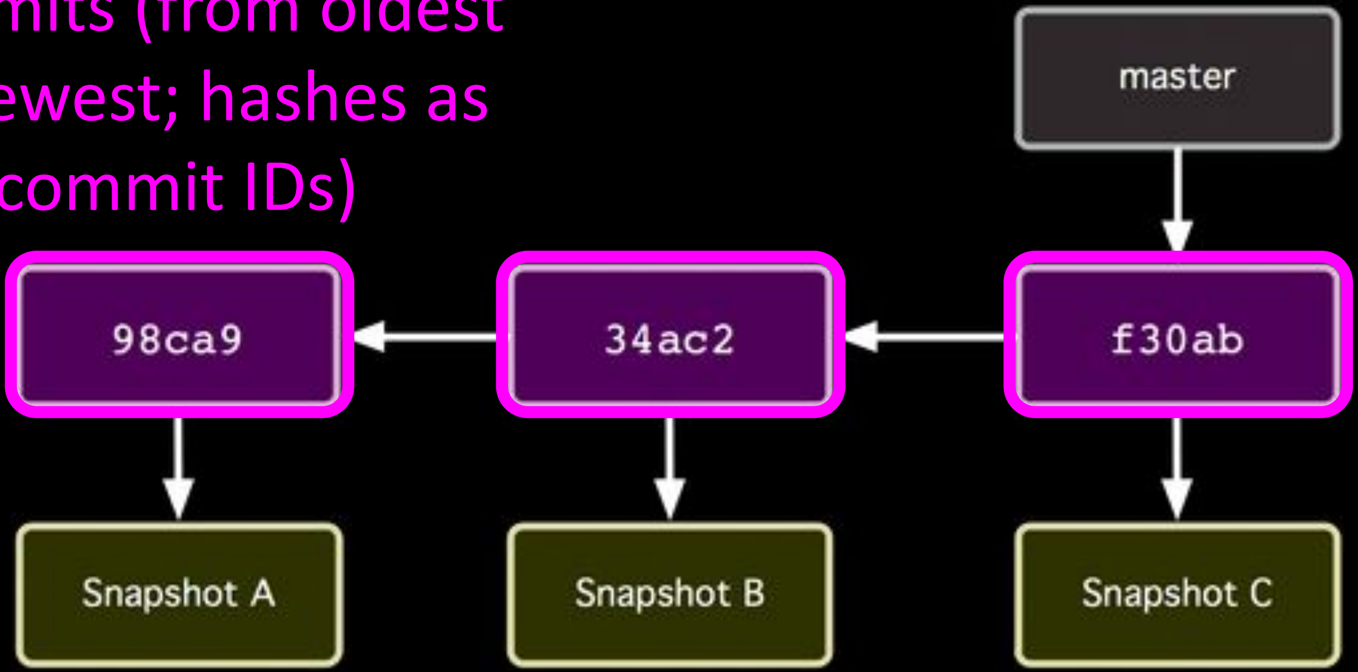
How the repos is organized



http://git-scm.com/book/

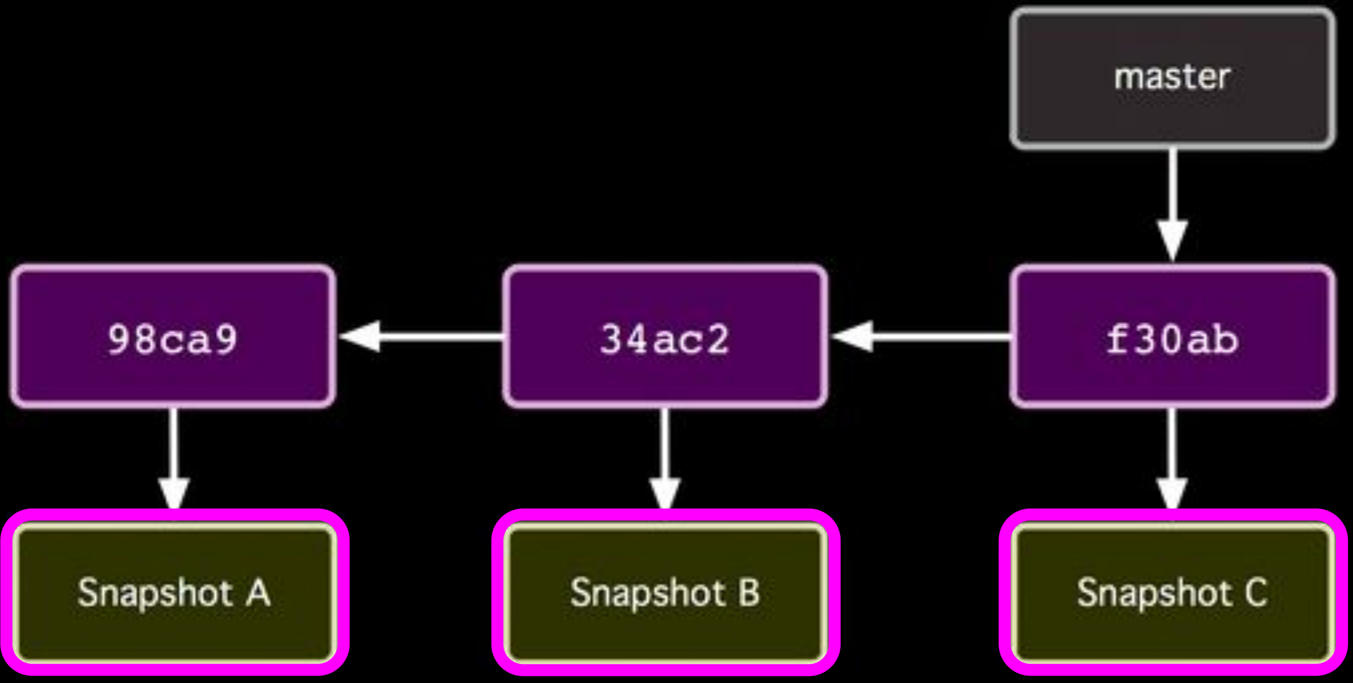
How the repos is organized

Commits (from oldest to newest; hashes as commit IDs)



http://git-scm.com/book/

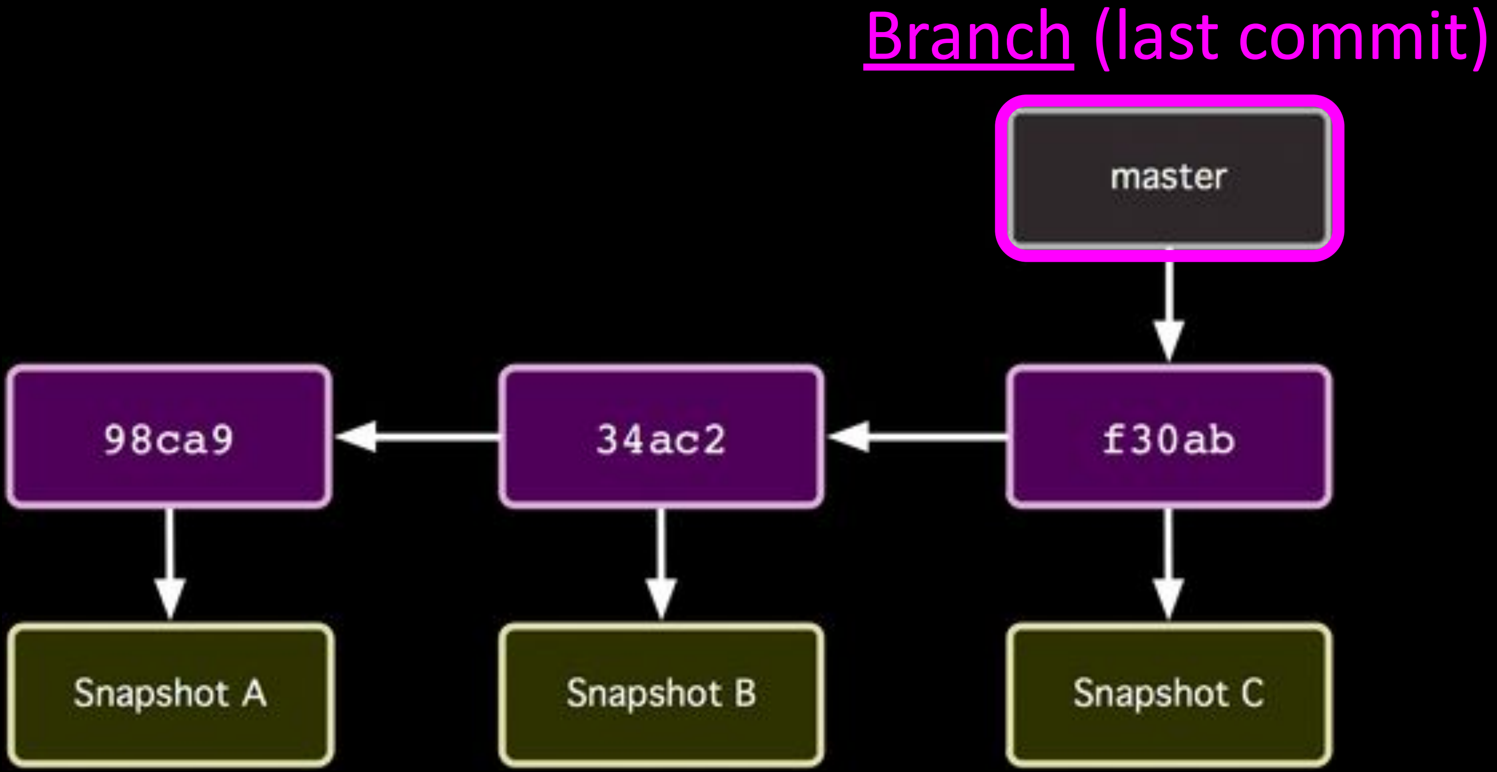
How the repos is organized



Snapshot of all files
at each commit

http://git-scm.com/book/

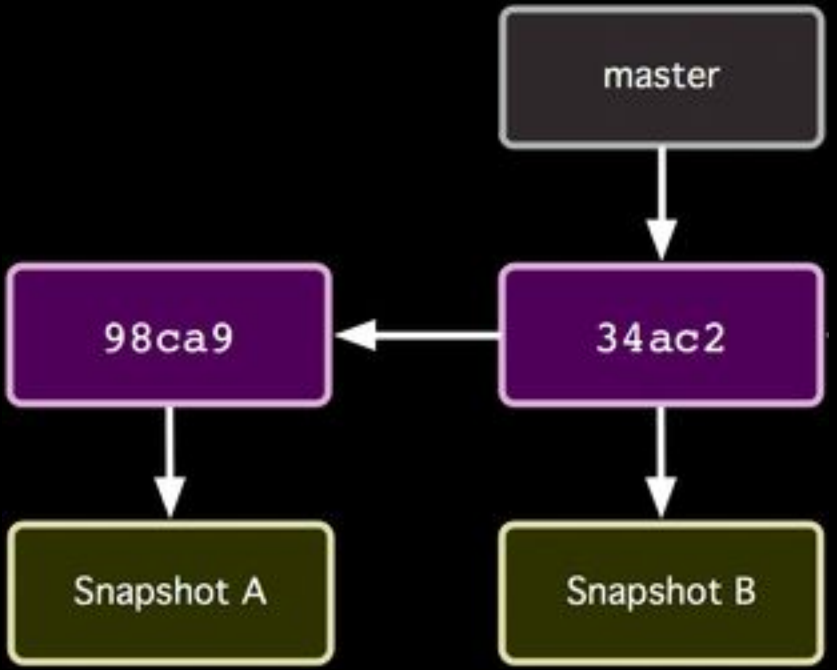
How the repos is organized



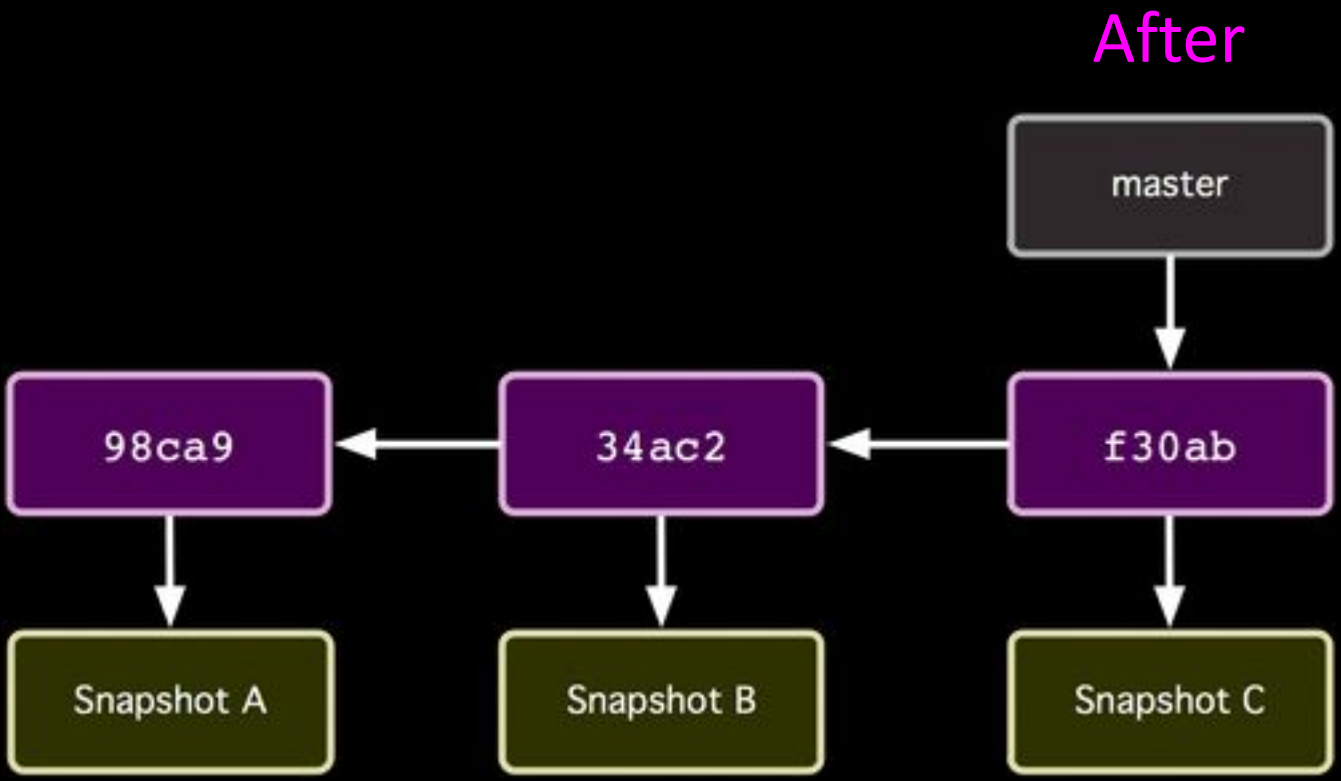
http://git-scm.com/book/

How commit works

Before



How commit works



http://git-scm.com/book/

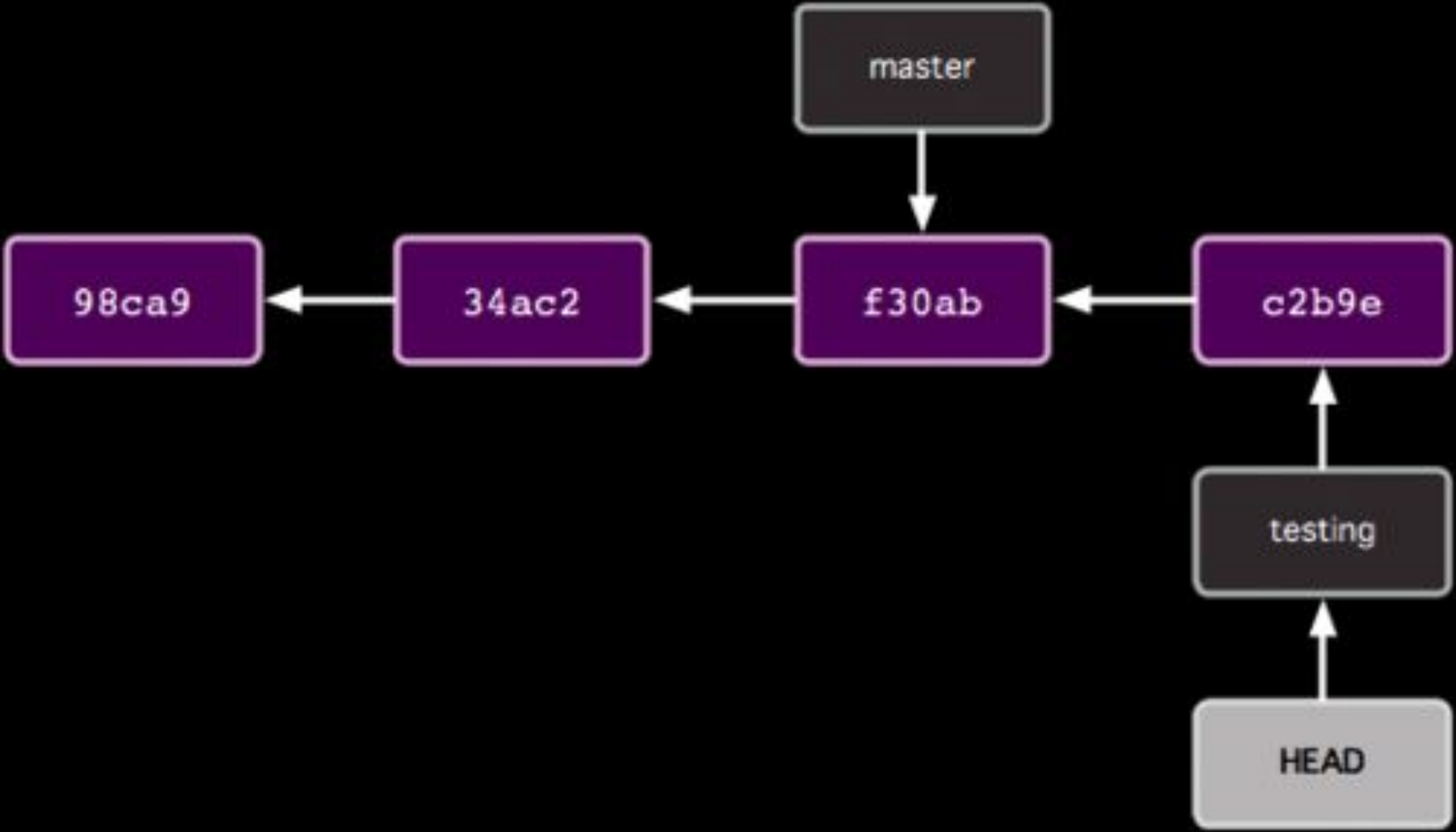
Common Workflow

1. Create temp local branch
2. Checkout temp branch
3. Edit/Add/Commit on temp branch
4. Checkout master branch
5. Pull to update master branch
6. Merge temp branch with updated master
7. Delete temp branch
8. Push to update server repos

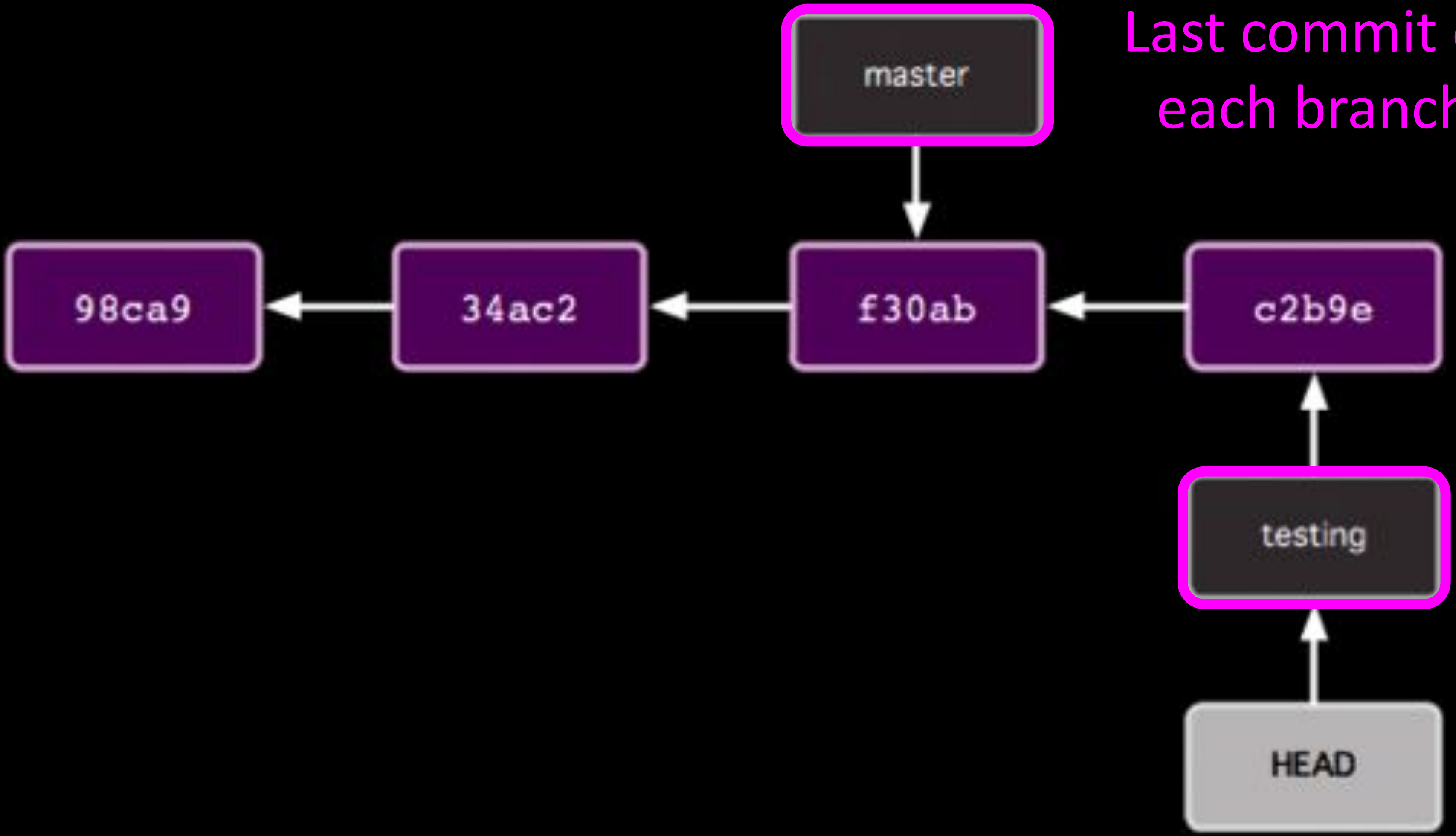
Make changes in local branch

Merge with GitHub repos

Organization with two branches

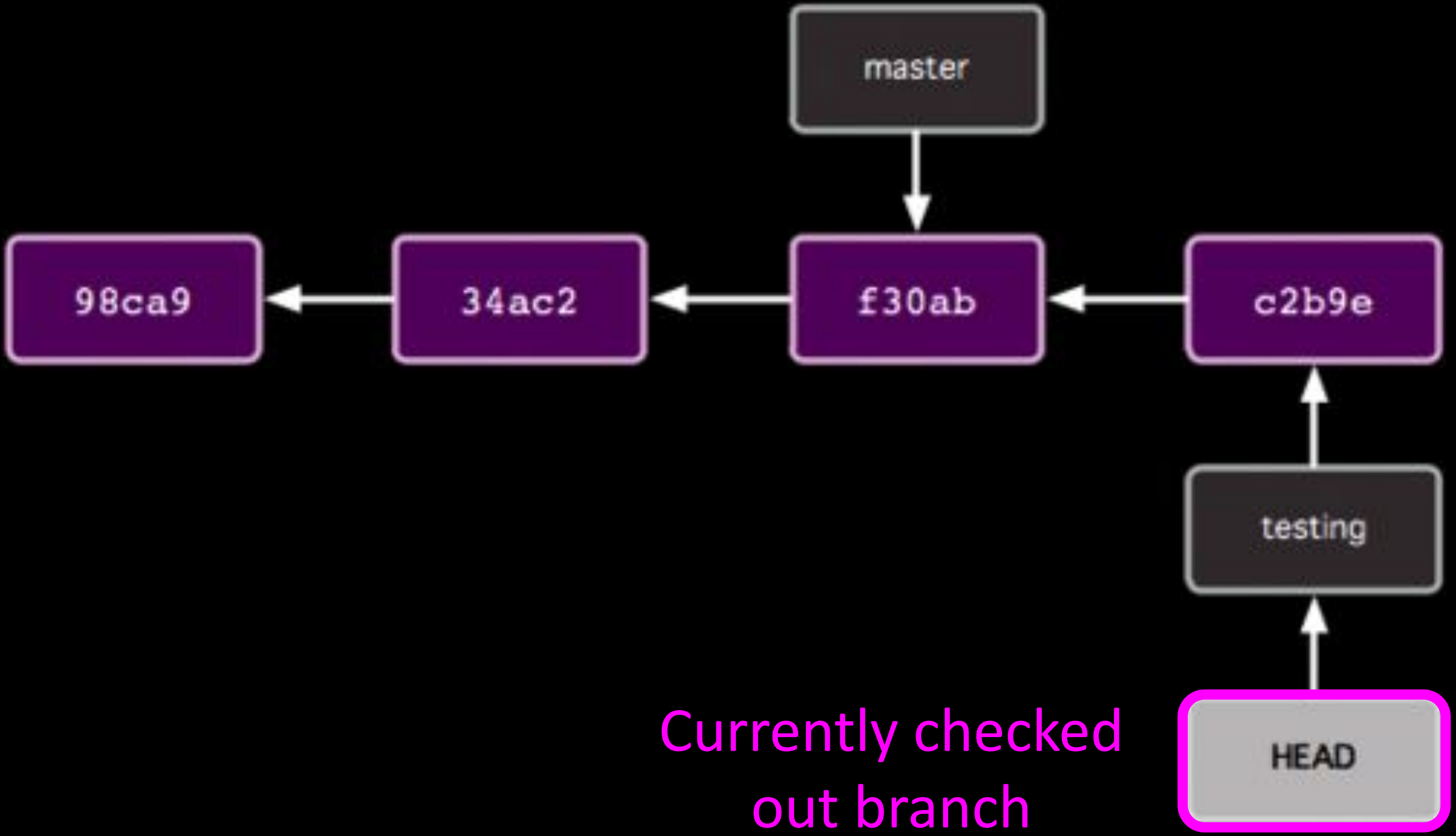


Organization with two branches



Last commit of each branch

Organization with two branches



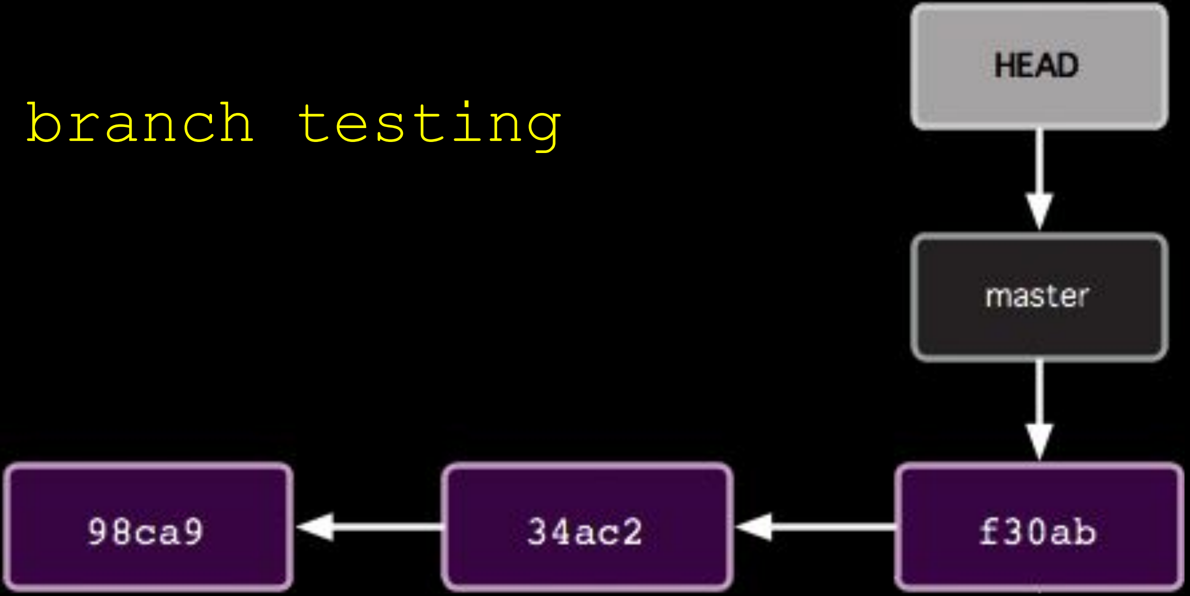
Currently checked out branch

Common Workflow

1. Create temp local branch
2. Checkout temp branch
3. Edit/Add/Commit on temp branch
4. Checkout master branch
5. Pull to update master branch
6. Merge temp branch with updated master
7. Delete temp branch
8. Push to update server repos

How git branch works

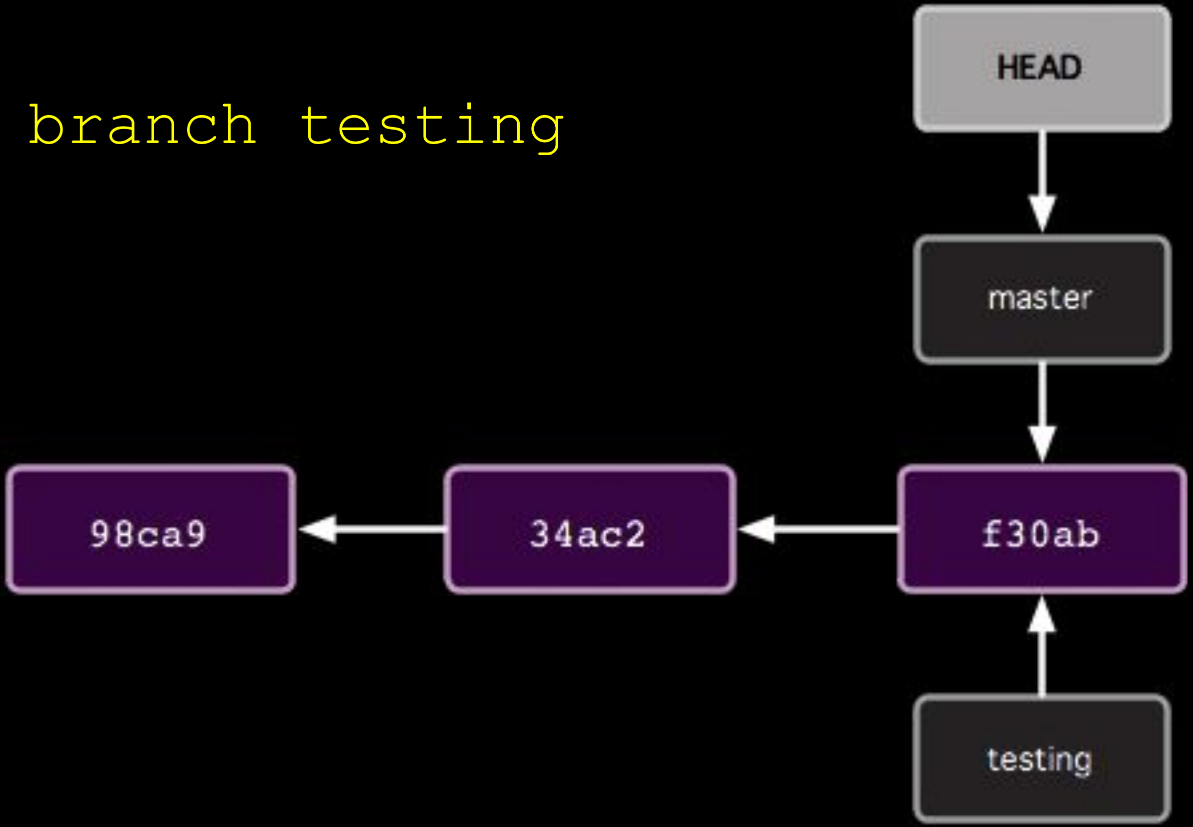
```
$ git branch testing
```



Before

How git branch works

```
$ git branch testing
```



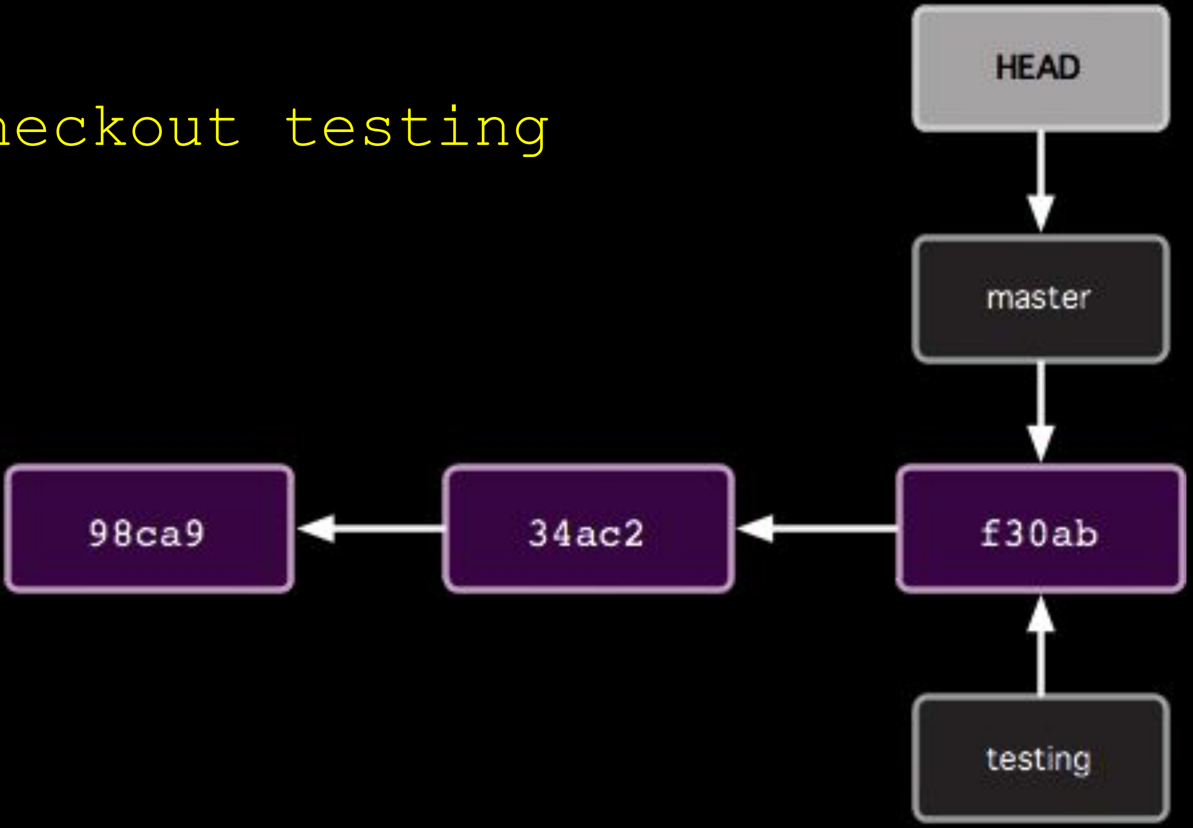
After

Common Workflow

1. Create temp local branch
2. Checkout temp branch
3. Edit/Add/Commit on temp branch
4. Checkout master branch
5. Pull to update master branch
6. Merge temp branch with updated master
7. Delete temp branch
8. Push to update server repos

How git checkout works

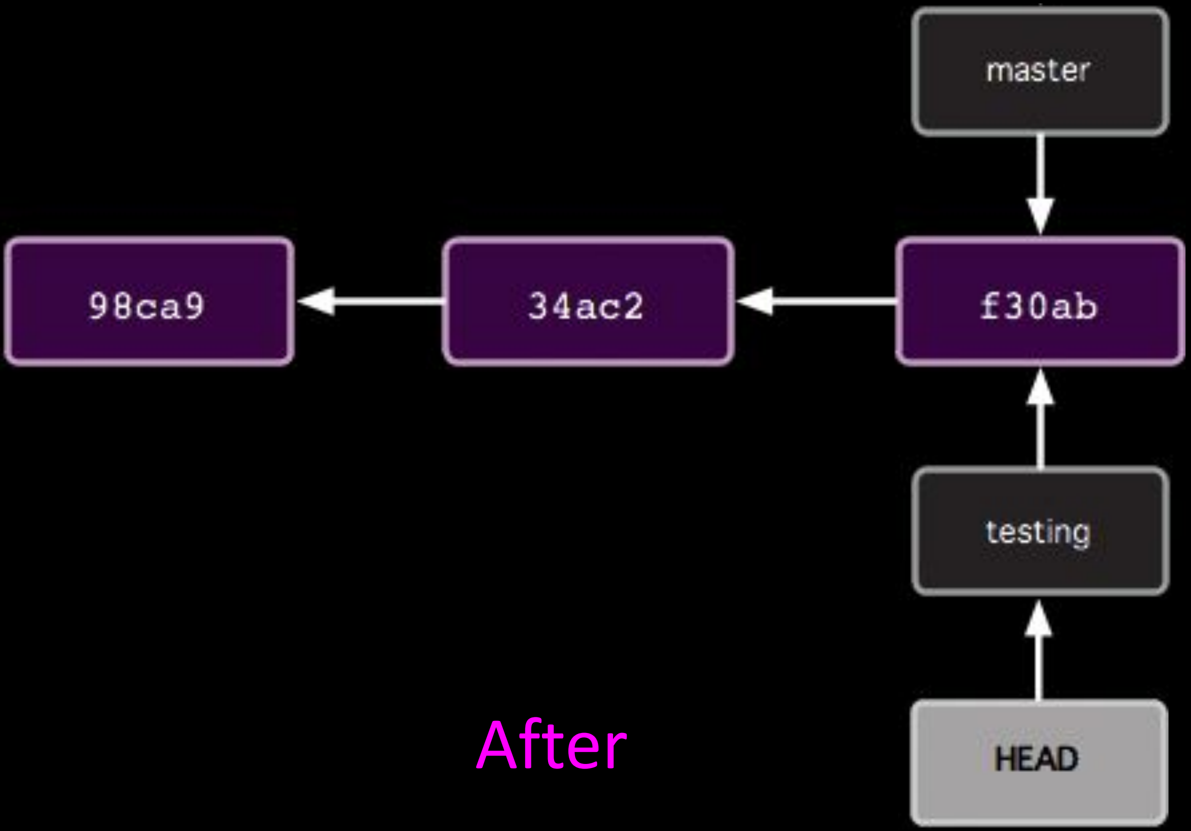
```
$ git checkout testing
```



Before

How git checkout works

```
$ git checkout testing
```



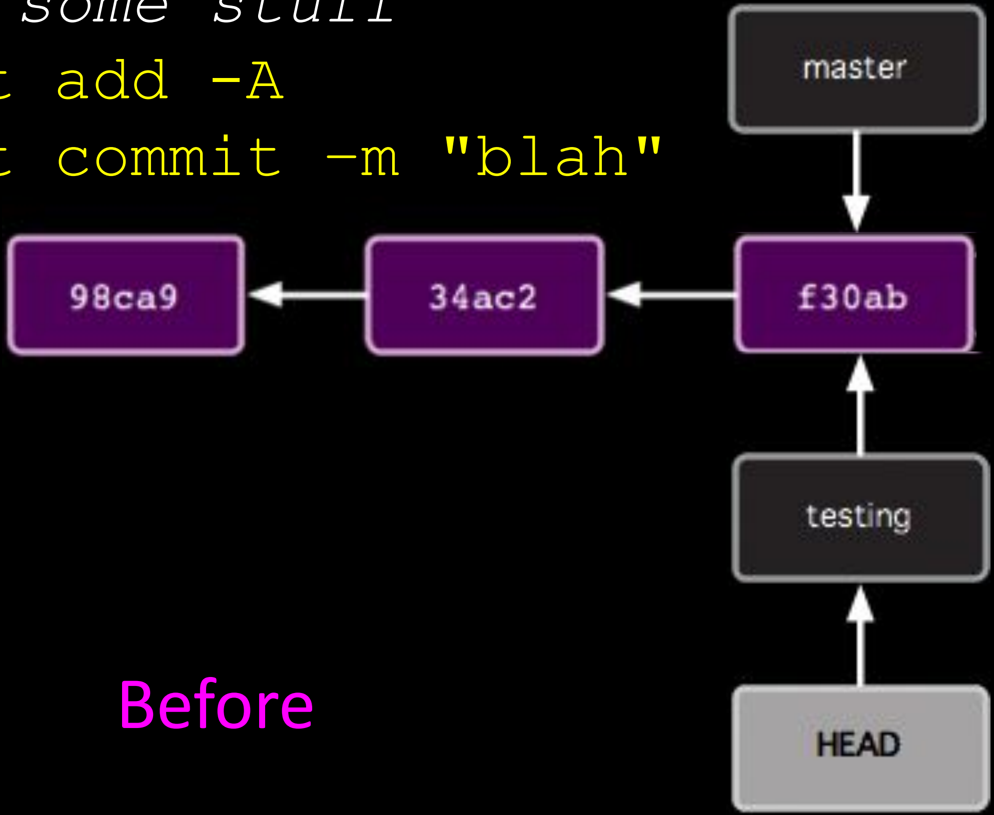
After

Common Workflow

1. Create temp local branch
2. Checkout temp branch
3. Edit/Add/Commit on temp branch
4. Checkout master branch
5. Pull to update master branch
6. Merge temp branch with updated master
7. Delete temp branch
8. Push to update server repos

How git commit works with multiple branches

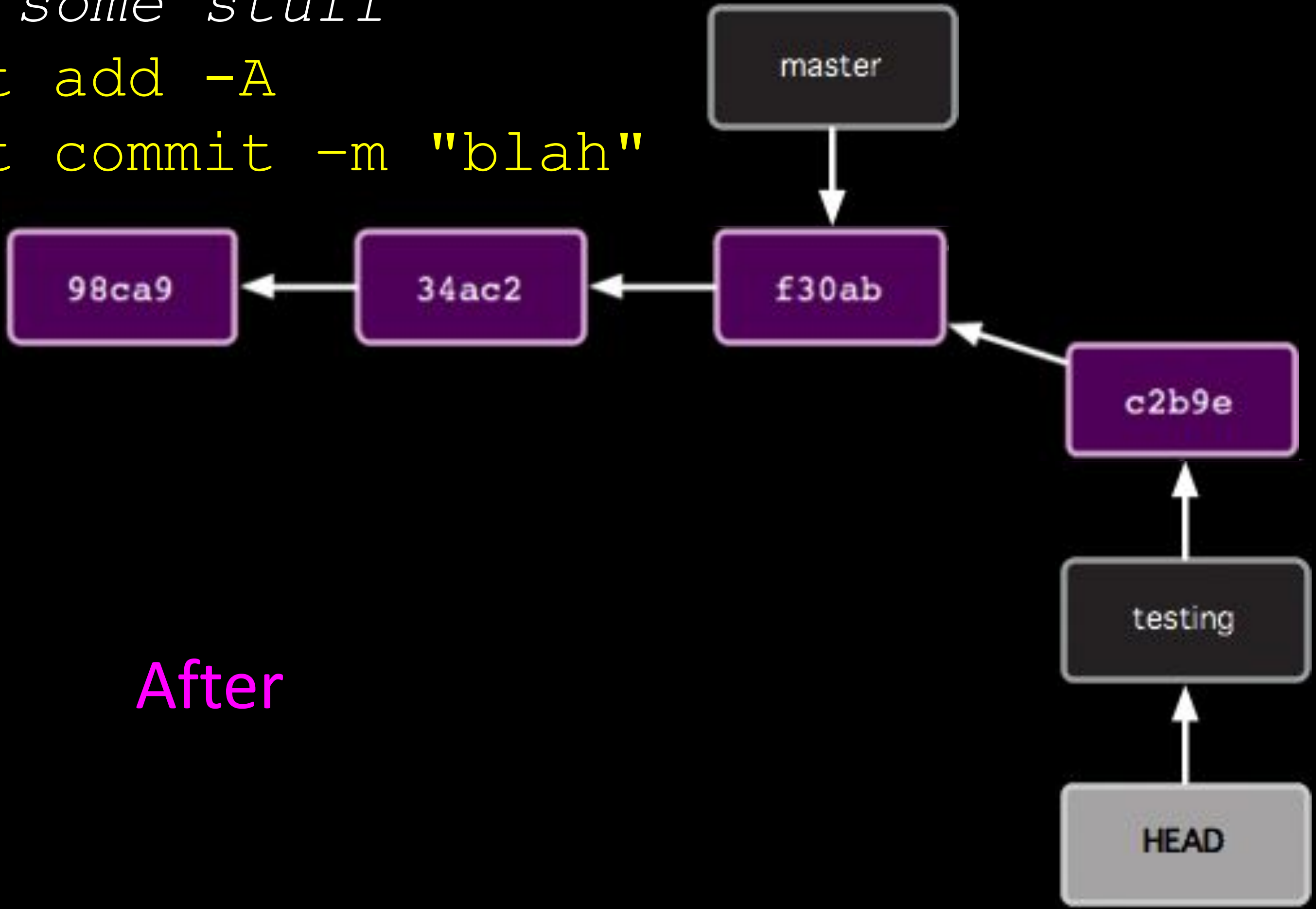
```
Edit some stuff
$ git add -A
$ git commit -m "blah"
```



Before

How git commit works with multiple branches

```
Edit some stuff
$ git add -A
$ git commit -m "blah"
```



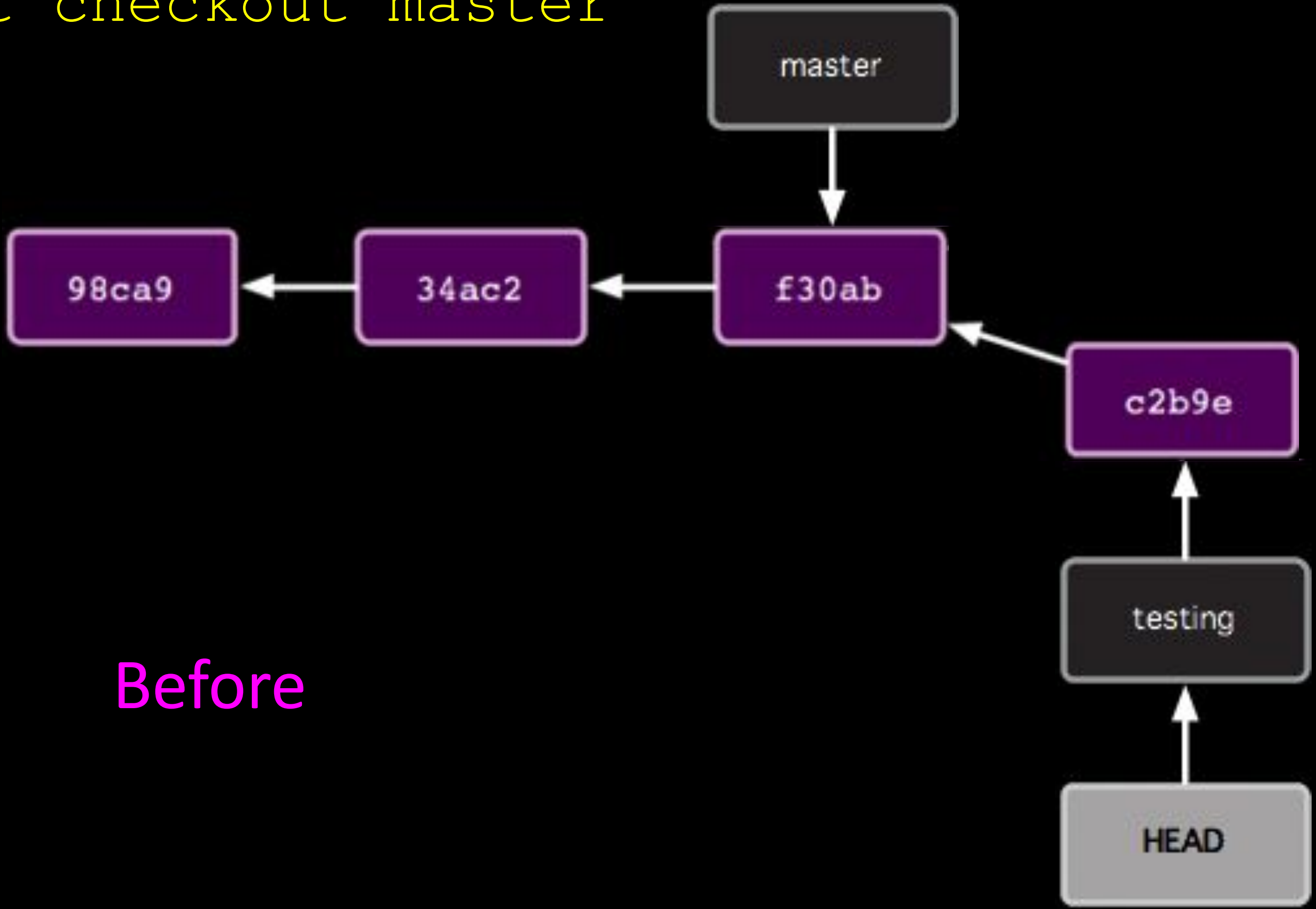
After

Common Workflow

1. Create temp local branch
2. Checkout temp branch
3. Edit/Add/Commit on temp branch
4. Checkout master branch
5. Pull to update master branch
6. Merge temp branch with updated master
7. Delete temp branch
8. Push to update server repos

How git checkout works

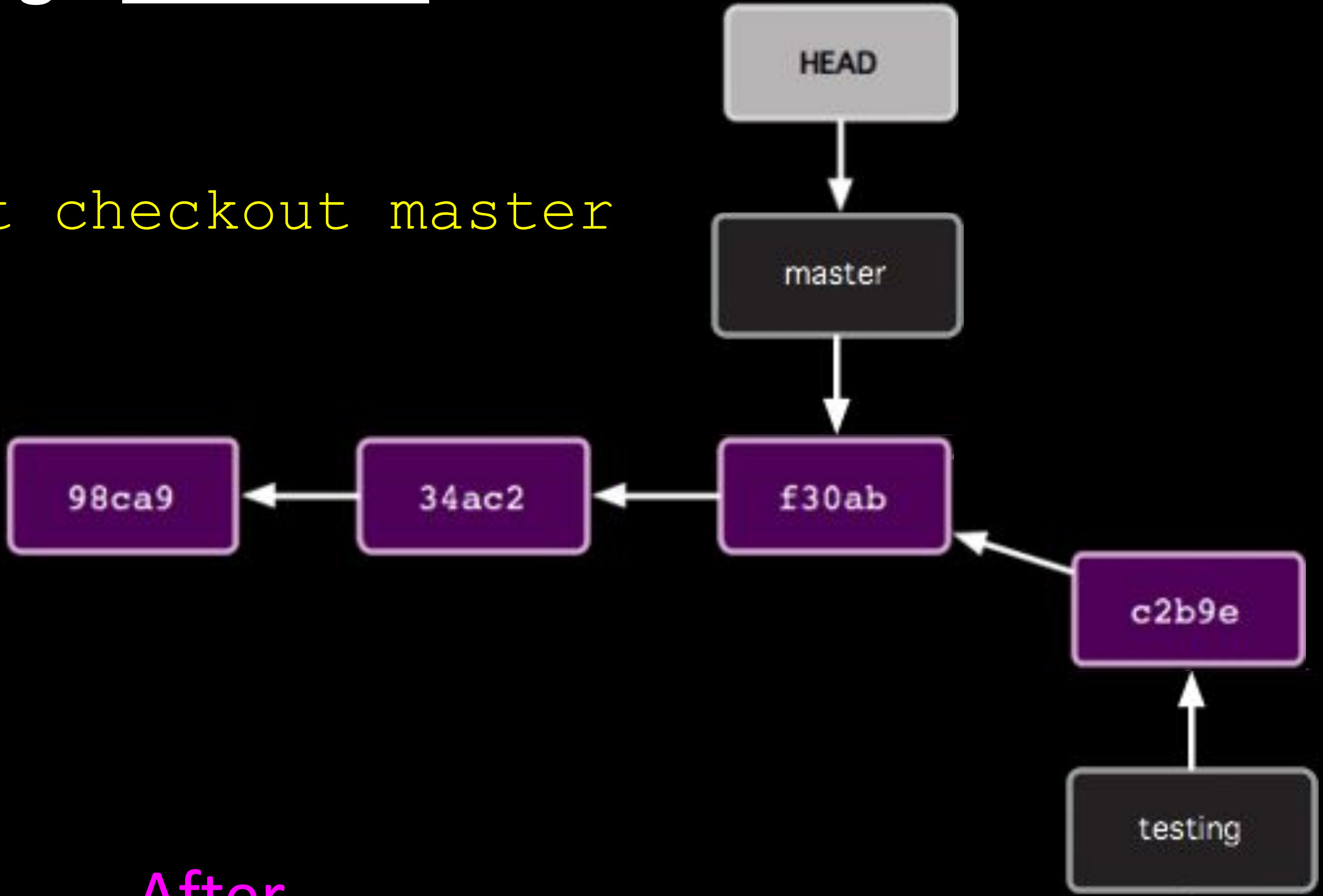
```
$ git checkout master
```



Before

How git checkout works

```
$ git checkout master
```



After

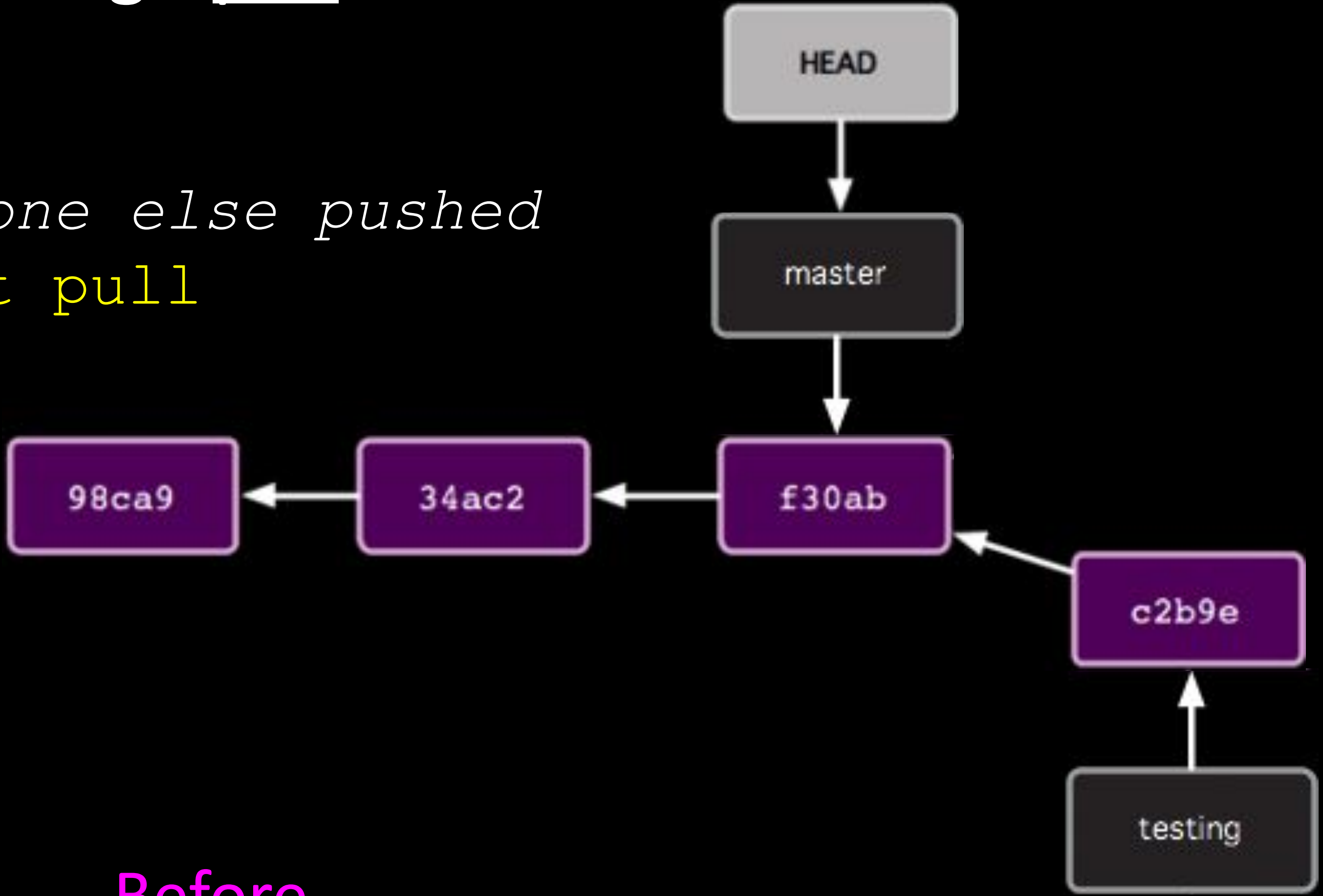
Common Workflow

1. Create temp local branch
2. Checkout temp branch
3. Edit/Add/Commit on temp branch
4. Checkout master branch
5. Pull to update master branch
6. Merge temp branch with updated master
7. Delete temp branch
8. Push to update server repos

How git pull works

Someone else pushed

```
$ git pull
```

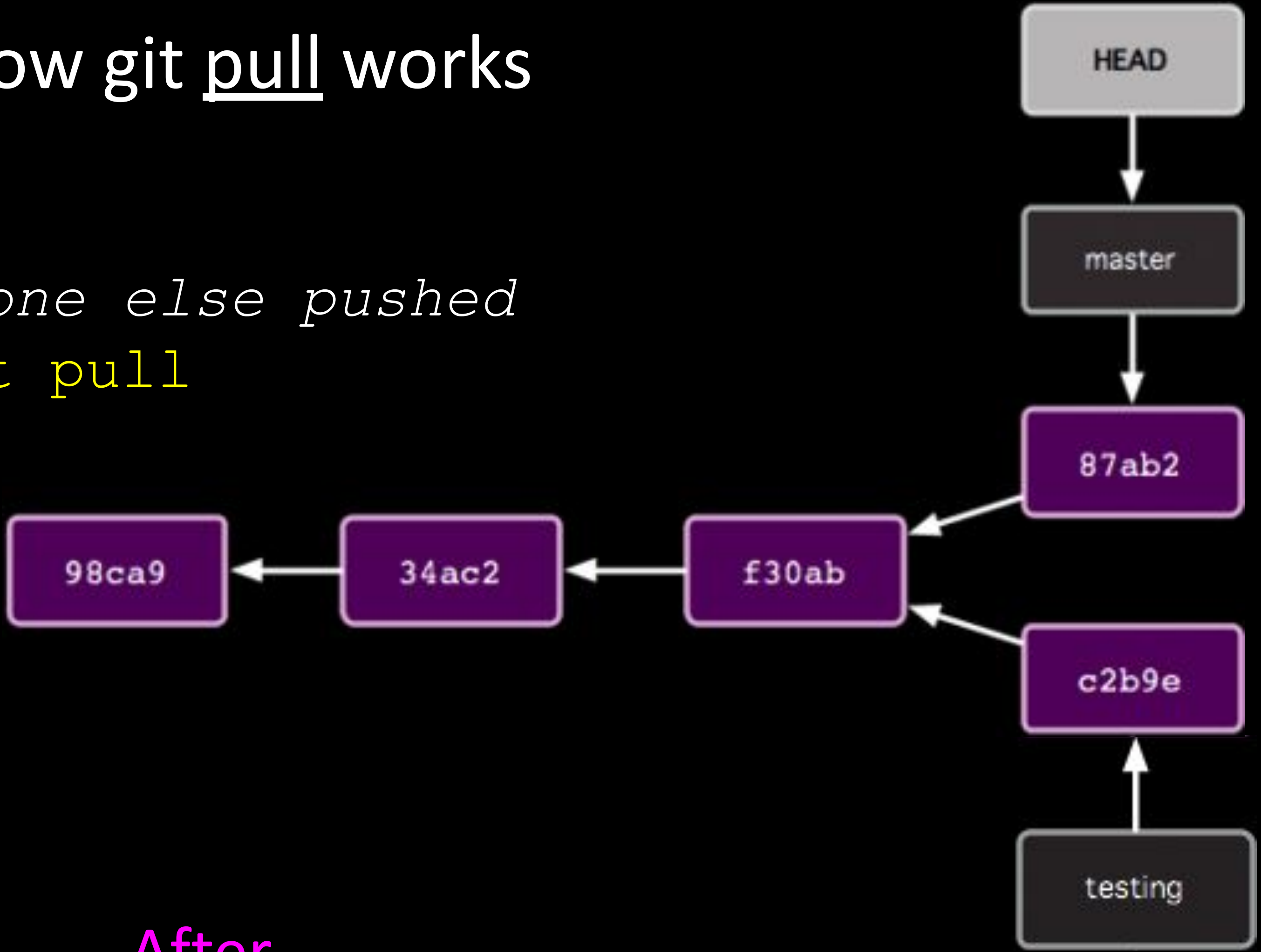


Before

How git pull works

Someone else pushed

```
$ git pull
```



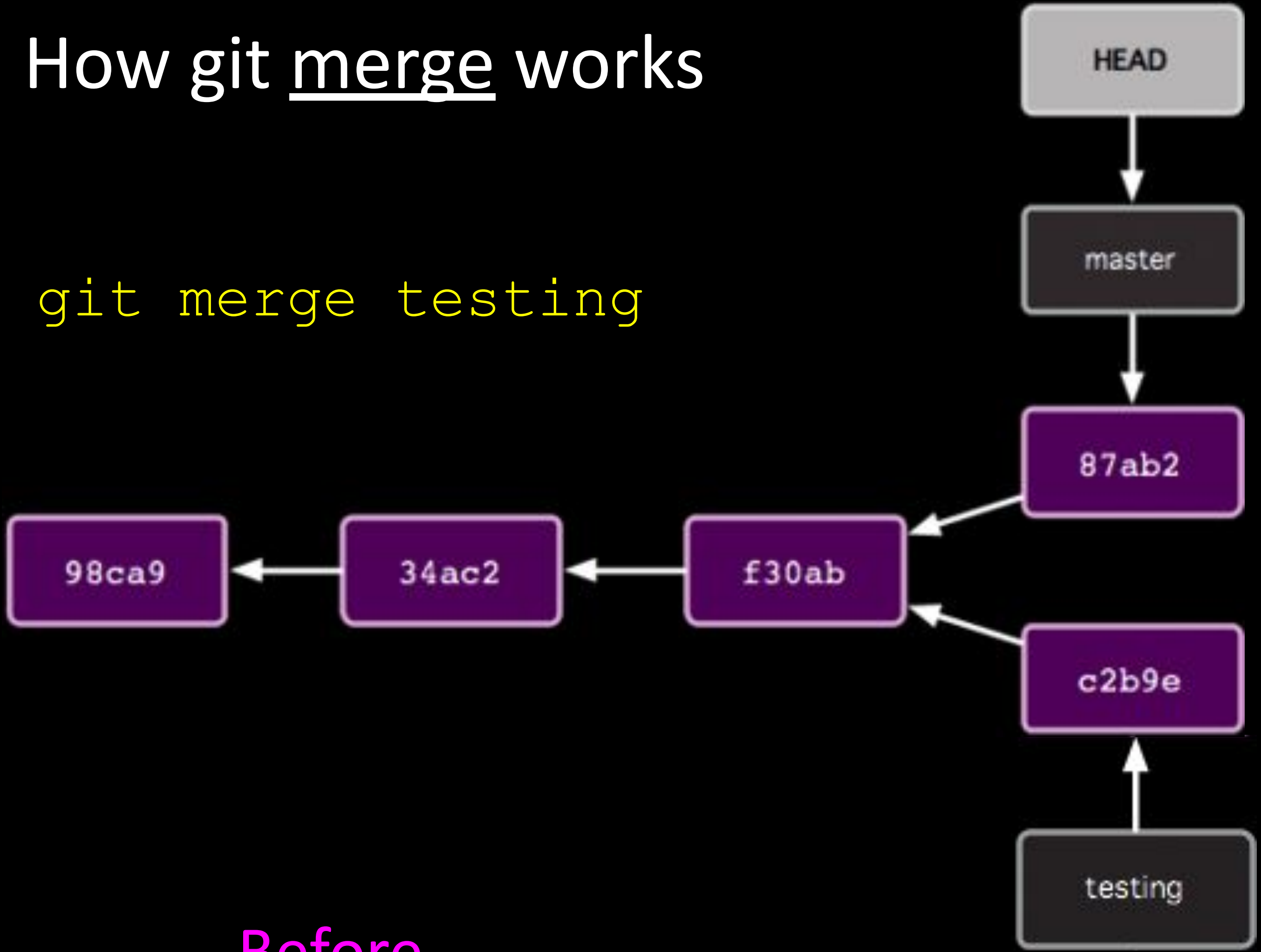
After

Common Workflow

1. Create temp local branch
2. Checkout temp branch
3. Edit/Add/Commit on temp branch
4. Checkout master branch
5. Pull to update master branch
6. Merge temp branch with updated master
7. Delete temp branch
8. Push to update server repos

How git merge works

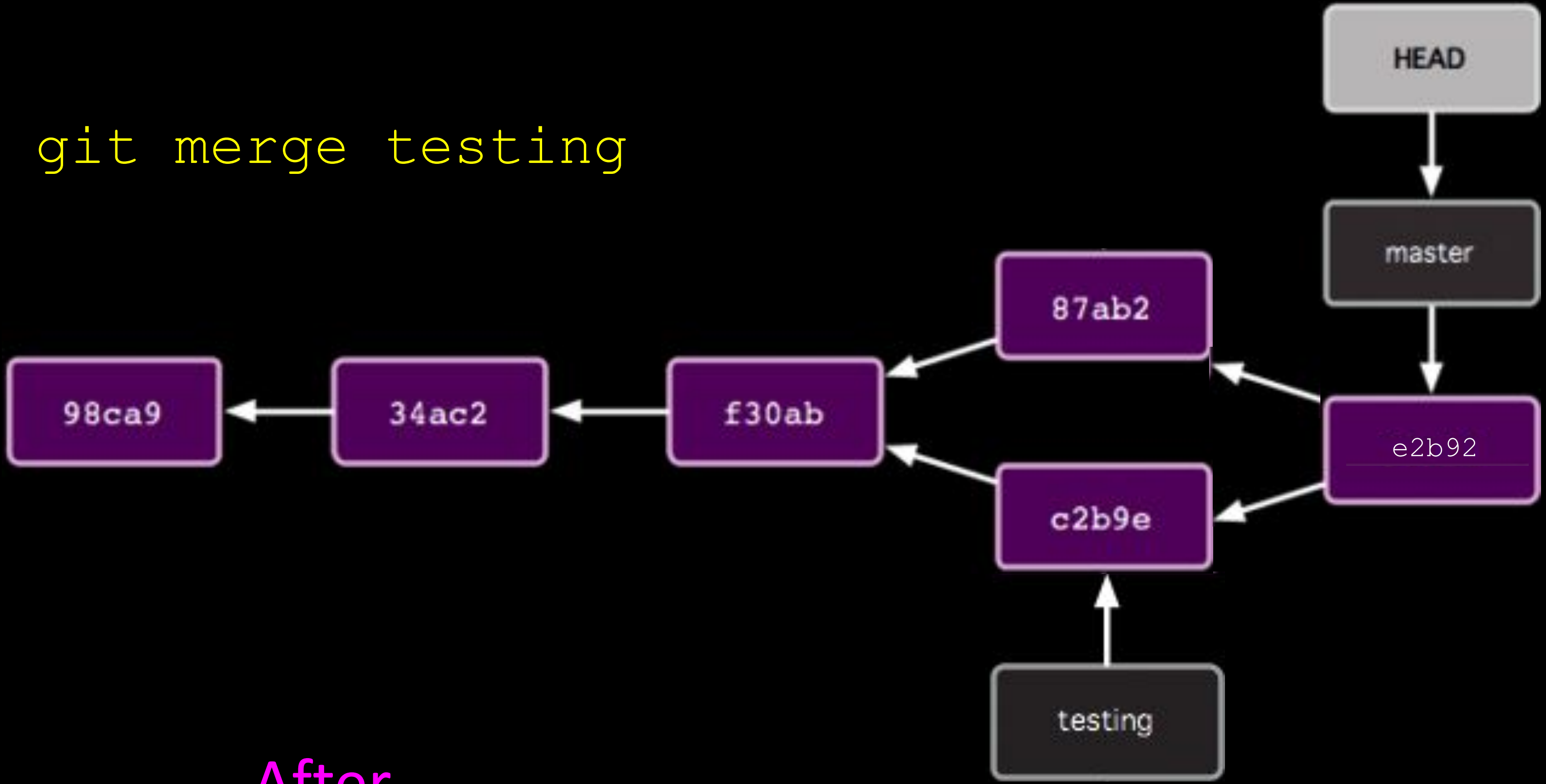
```
$ git merge testing
```



Before

How git merge works

```
$ git merge testing
```



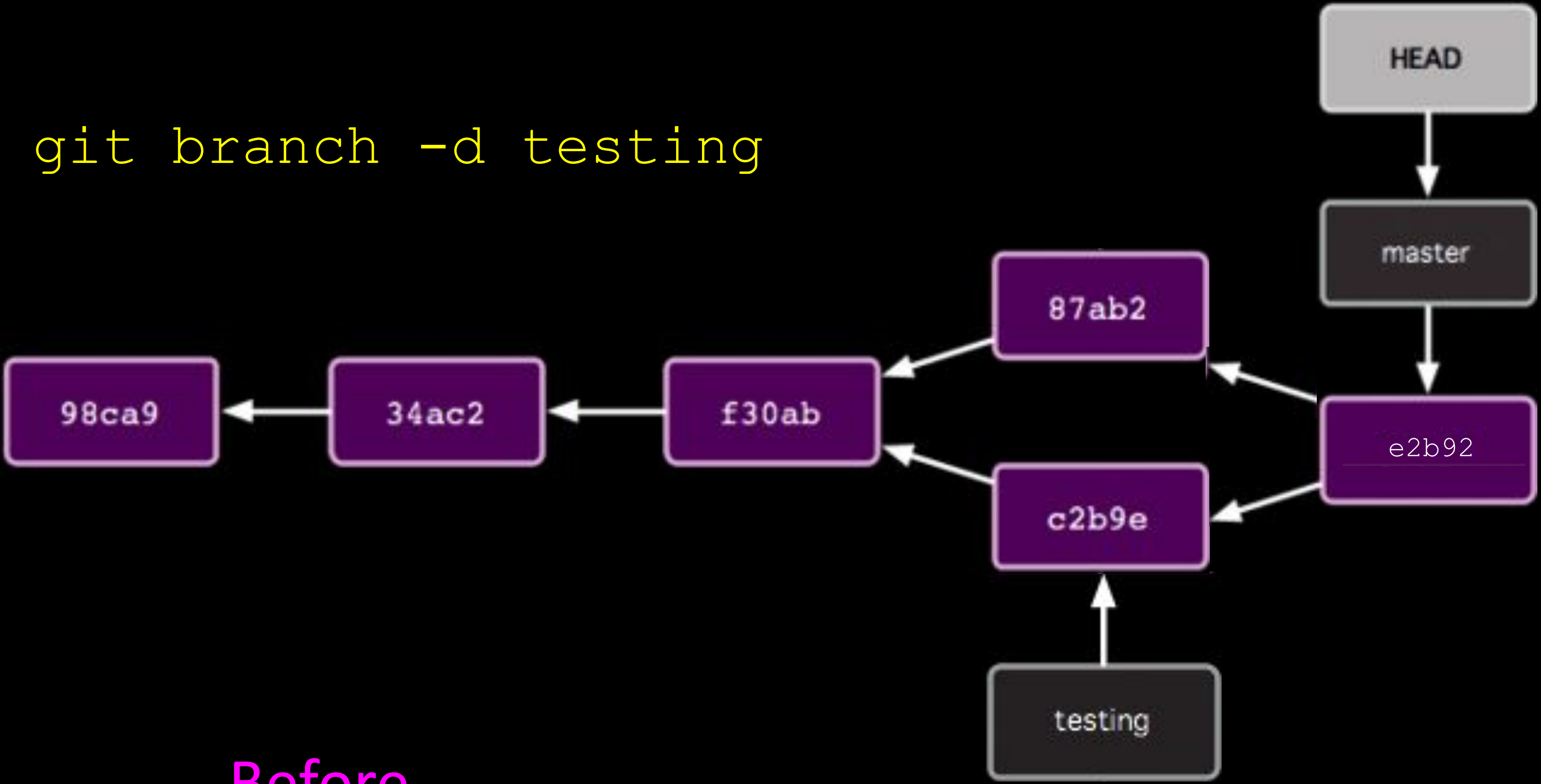
After

Common Workflow

1. Create temp local branch
2. Checkout temp branch
3. Edit/Add/Commit on temp branch
4. Checkout master branch
5. Pull to update master branch
6. Merge temp branch with updated master
7. Delete temp branch
8. Push to update server repos

How to delete branches

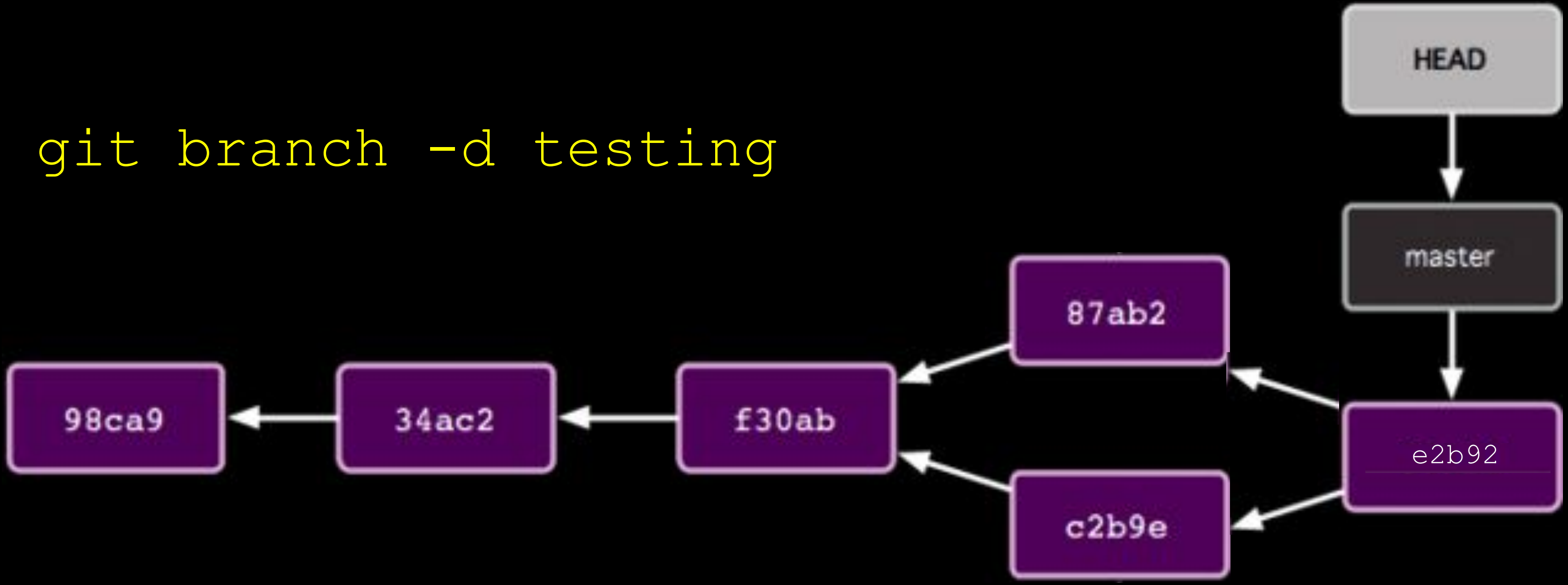
```
$ git branch -d testing
```



Before

How to delete branches

```
$ git branch -d testing
```



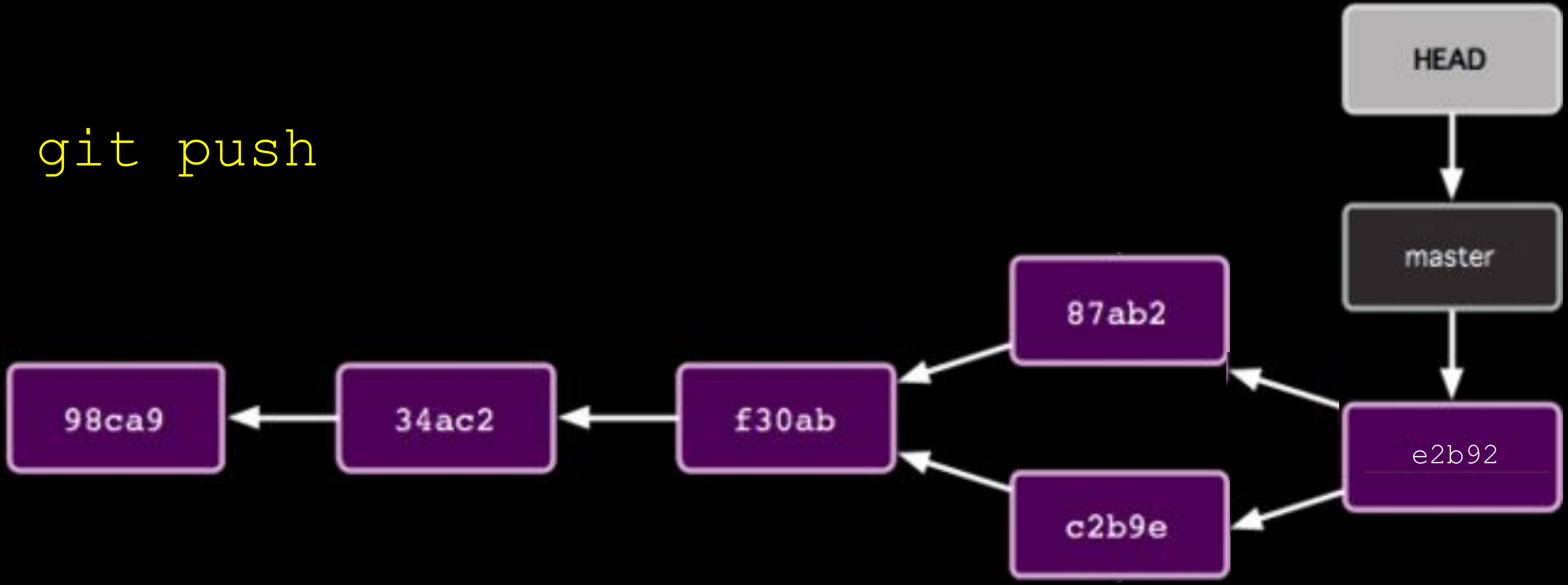
After

Common Workflow

1. Create temp local branch
2. Checkout temp branch
3. Edit/Add/Commit on temp branch
4. Checkout master branch
5. Pull to update master branch
6. Merge temp branch with updated master
7. Delete temp branch
8. Push to update server repos

How git push works

```
$ git push
```



Should update server repos
(if no one else has pushed commits to master branch since last pull)

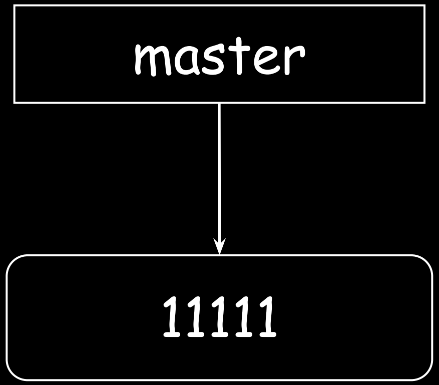
Tips

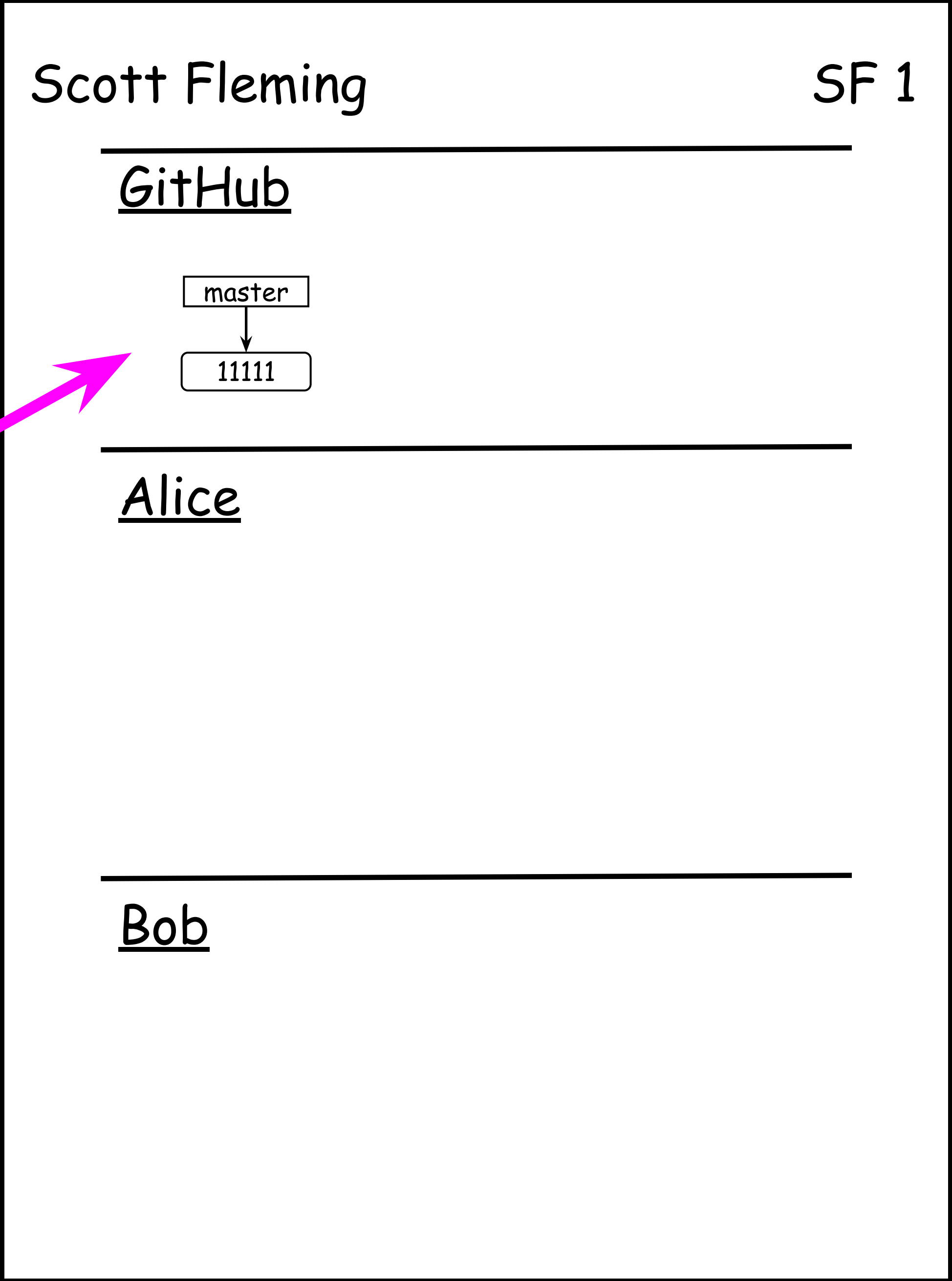
- git output contains lots of hints
  - git status is your friend!
- Merging may not be as easy as I showed
  - E.g.: Multiple collabs updated same parts of file
  - See Pro Git 3.2
- Pull before starting temp branch
- Team communication important!

Pop Quiz

- 5 questions
- Update diagram in each
  - Commit nodes
  - Branch nodes
- Based on actions of Alice and Bob
  - Collaborating via GitHub repo

Start like this





Question 1

- Alice:
  - $ git clone https://github.com/whatever.git
  - $ cd whatever

- Bob:
  - $ git clone https://github.com/whatever.git
  - $ cd whatever

(include the HEAD node)

Question 2

- Alice:
  - $ git branch myfix
  - $ git checkout myfix
- (Alternatively)
  - $ git checkout -b myfix

Question 3

- Alice:
  - $ rails generate scaffold User …
  - $ git add -A
  - $ git commit -m "Added User" # 22222
- Bob:
  - $ rails generate scaffold Micropost …
  - $ git add -A
  - $ git commit -m "Added Micropost" # 33333

Question 4

- Bob:
  - git push

Question 5

- Alice:
  - git pull

Appendix

What if…

Alice did this:

app/models/micropost.rb

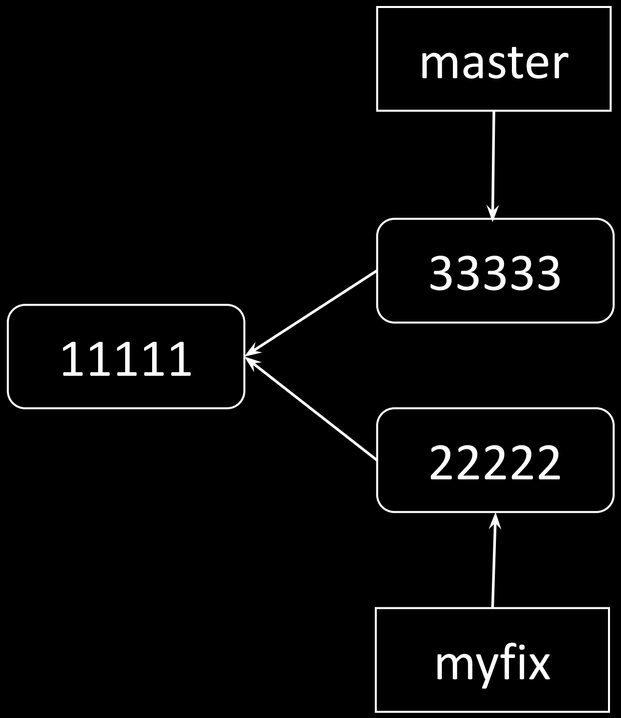
```
class Micropost < ActiveRecord::Base
  validates :content, length: { maximum: 140 }
end
```

Bob did this:

app/models/micropost.rb

```
class Micropost < ActiveRecord::Base
  validates :content, length: { maximum: 120 }
end
```

What if Alice did this?



```
$ git checkout master
$ git merge myfix
$ git merge myfix
Auto-merging app/models/micropost.rb
Automatic merge failed; fix conflict and then commit result.
```

app/models/micropost.rb

```
class Micropost < ActiveRecord::Base
<<<<<<< HEAD
  validates :content, length: { maximum: 140 }
=======
  validates :content, length: { maximum: 120 }
>>>>>>> myfix
end
```

To resolve:

Manually fix the file; git add and commit

QUESTIONS?
CONCERNS?
COMMENTS?