Git

Git ≠ Github

git ≠ GitHub

What is Version Control?

```
Aaron@HELIOS ~/112_term_project
$ ls
termproject_actually_final  termproject_v10  termproject_v3
termproject_final           termproject_v11  termproject_v4
termproject_handin          termproject_v12  termproject_v5
termproject_old_idea        termproject_v13  termproject_v6
termproject_superfrogger    termproject_v14  termproject_v7
termproject_temp            termproject_v15  termproject_v8
termproject_this_one_works  termproject_v16  termproject_v9
termproject_v1              termproject_v2
```

Named Folders Approach

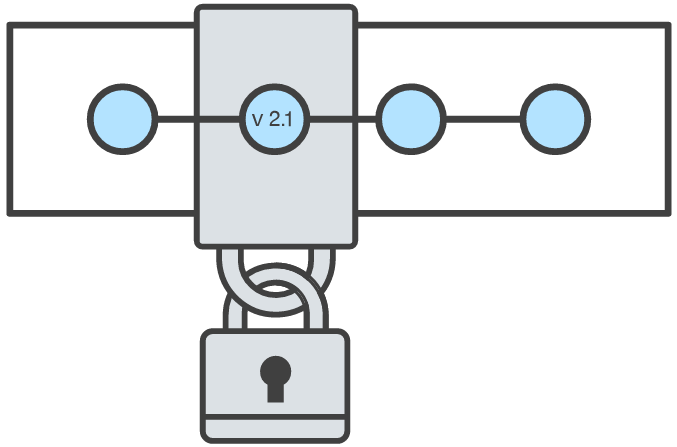
```
Aaron@HELIOS ~/112_term_project
$ ls
termproject_actually_final  termproject_v10  termproject_v3
termproject_final           termproject_v11  termproject_v4
termproject_handin          termproject_v12  termproject_v5
termproject_old_idea        termproject_v13  termproject_v6
termproject_superfrogger    termproject_v14  termproject_v7
termproject_temp            termproject_v15  termproject_v8
termproject_this_one_works  termproject_v16  termproject_v9
termproject_v1              termproject_v2
```

- Easy
- Familiar
- …

- Can be hard to track
- Memory-intensive
- Can be slow
- Hard to share
- No record of authorship

Goals of Version Control

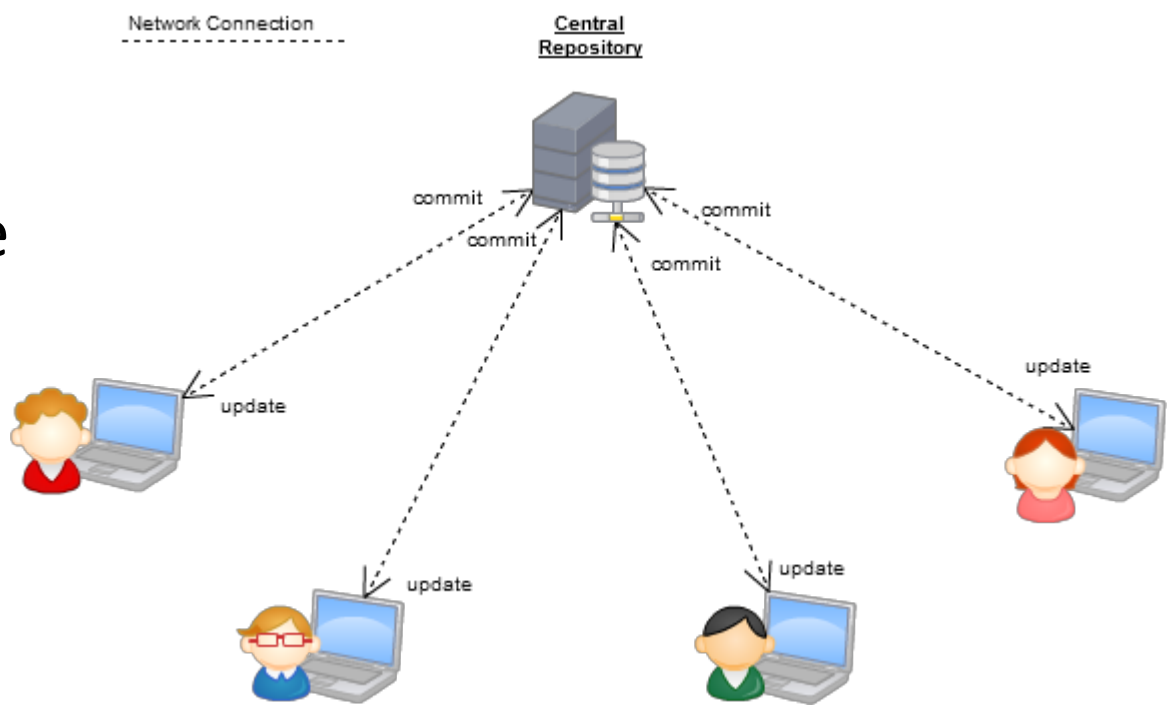
- Be able to search through revision history and retrieve previous versions of any file in a project
- Be able to share changes with collaborators on a project
- Be able to confidently make large changes to existing files



https://www.atlassian.com/git/tutorials/what-is-version-control

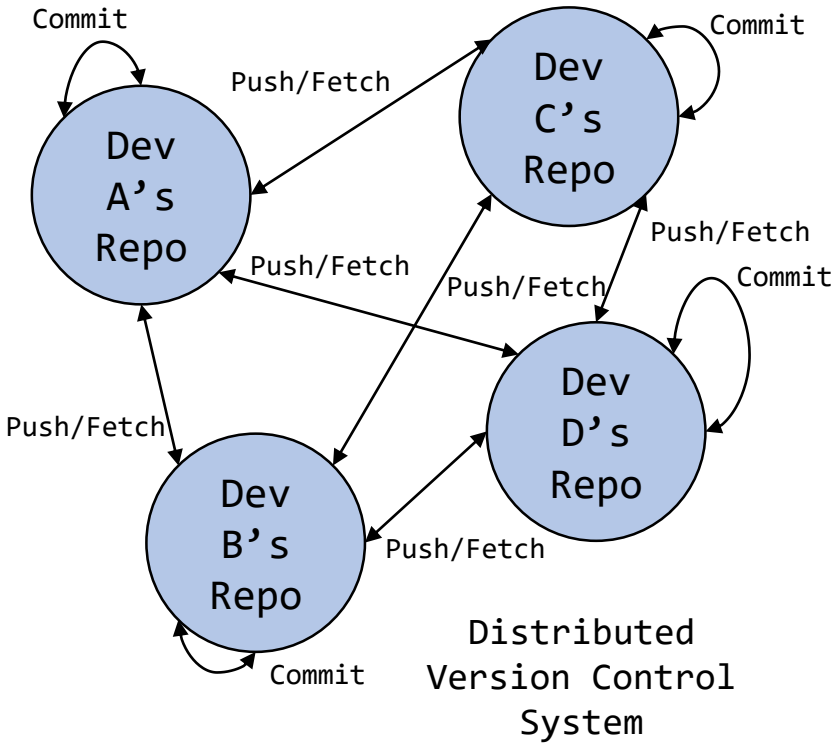
Centralized Version Control Systems

- A central repository determines the **order** of versions of the project
- Collaborators “push” changes to the files to the repository
- Any new changes must be compatible with the most recent version of the repository. If it isn’t, somebody must “merge” it in.
- Examples: SVN, CVS, Perforce



Distributed Version Control Systems (DVCS)

- No central repository, each developer has their own copy
- Developers work on their own copy of the repository locally and sync changes with others
- Examples: Git, Mercurial



Git

- Created in 2005 by Linus Torvalds to maintain the Linux kernel. Oh, and he created that too.
- Distributed VCS

https://www.git-scm.com/

Installing Git

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Git Init

Initializes a new git repository in an existing folder

- The folder is now called a git **repository**
- Changes to any files in the folder (and its subfolders) can be tracked by git
- Git stores its metadata in a hidden .git folder in the repository root

```
$ mkdir myrepo
$ cd myrepo
$ git init
```

Git Clone

- Download an existing repository (and all of its history!)

```
$ git clone https://github.com/autolab/Autolab.git
$ cd Autolab
```

The .git folder

- Every git repository has a .git directory in the toplevel project directory
- This is where all git commit objects and metadata are stored
- **Don't delete it!** Doing so deletes the repository
- Folders starting with a dot are hidden on UNIX

```
Aaron@HELIOS ~/Dropbox/Dropbox Documents/98174/www (master)
$ ls -a
.  ..  .git  css  f16  homework  index.html  lecturenotes  slides

Aaron@HELIOS ~/Dropbox/Dropbox Documents/98174/www (master)
$ ls .git
COMMIT_EDITMSG  config       hooks  info  objects      refs
HEAD            description  index  logs  packed-refs
```

Git Log

List the history of a repository

```
$ git log
```

Press ‘q’ to exit, use arrow keys (or j,k) to scroll

```
commit fad72e4a28f84f004718e57bfa3b7e21c8f4f8cf
Author: Devansh Kukreja <devanshkukreja1@gmail.com>
Date:   Sun Jul 30 12:54:33 2017 -0400

    Fixed Add-To-Slack btn

commit 38be9f4a3a79d0c7e2a724da0be53cb4f194975f
Author: Devansh Kukreja <devanshkukreja1@gmail.com>
Date:   Mon Jul 24 23:48:44 2017 -0400

    Updated main url

commit 630898d86ff4c6daf12c10ea08971f578995451e
Author: Devansh Kukreja <devanshkukreja1@gmail.com>
Date:   Mon Jul 24 23:46:26 2017 -0400

    Begun rollout of our Slack and updated logo

commit 0acbffcb74b16acdb802ac318834f6abbca97808
Author: Devansh Kukreja <devanshkukreja1@gmail.com>
Date:   Fri May 19 17:11:30 2017 -0400

    Remove disabled fields in create course user datum (#877)

commit e21a4186cf5d60755417bc6abff05f7884907730
Merge: 2159d9b 79678fe
Author: Aatish Nayak <aatishn@andrew.cmu.edu>
Date:   Thu May 4 01:49:47 2017 -0400

    Autolab Release v2.0.8

    Autolab Release v2.0.8

commit 79678fee0c7fed9aacb6614db5c031c0f2ac98c2
Merge: 73601bc 2159d9b
Author: Aatish Nayak <aatishn@andrew.cmu.edu>
Date:   Thu May 4 01:44:27 2017 -0400

    Merge branch 'master' into develop

commit 73601bc3fee518b613352839aa21bf16f5e57aee
Author: Jacob Buckheit <jbman223@gmail.com>
Date:   Thu May 4 01:42:48 2017 -0400

    Due date fix (#871)

commit 2159d9b2f6edad2684194ccbbef28c1632d58c72
Author: Aatish Nayak <aatishn@andrew.cmu.edu>
Date:   Tue May 2 12:11:35 2017 -0400

    Release v2.0.7 (#870)

    * Fixed Date time picker always displaying 12AM

    * move get child status to after moss command (#817)

    * Update .gitignore to ignore .vscode configs

    Namely address launch.json but also removes any additional vscode IDE-specific configurations

    * Hotfix tabs redirect (#824)

    * fixed problems page redirect

    * set up tab redirects by including a tag with each submit button
:|
```

What is `fad72e4`?

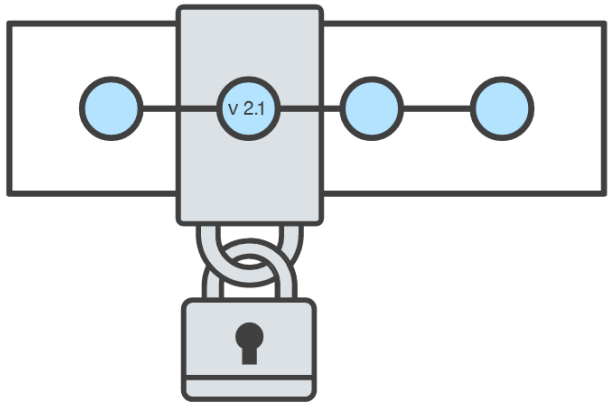
- Commits are uniquely represented by SHA-1 hashes
- The first 6-7 characters of a hash are usually enough to identify it uniquely from all the other commits in the repository
- This is called the **short hash**

Okay, so what is a commit?

1. A **snapshot** of all the files in a project at a particular time.
2. A **checkpoint** in your project you can come back to or refer to.

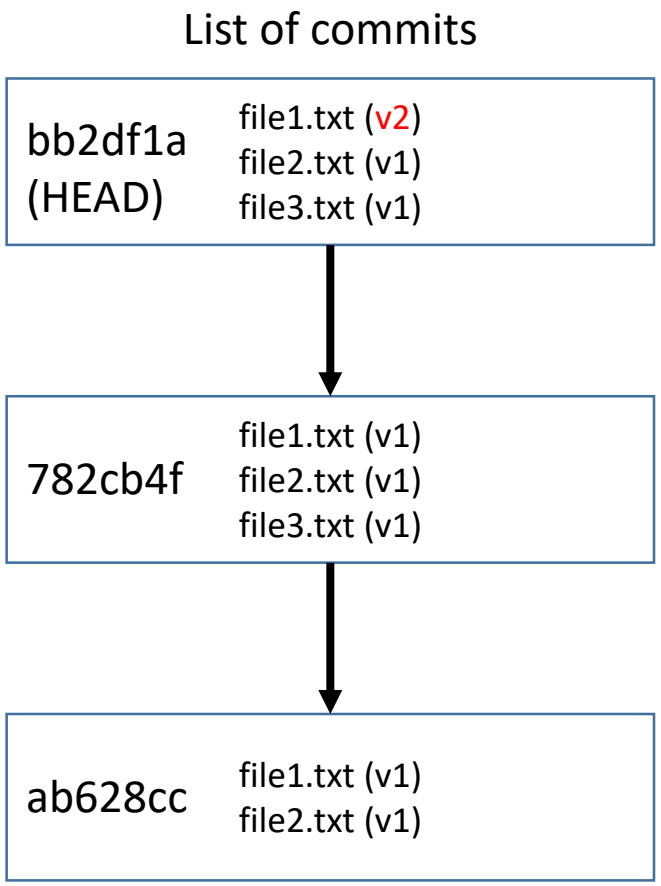
Anything else?

3. The **changes** a commit makes over the previous commit



Commits: Revisited

- Editing a file takes its state from 1 particular snapshot to the next
- When we edit a file, we can see it as a set of changes (a “diff”) from the snapshotted state of that file
- Commits bundle up sets of changes to a list of files



The Git Commit Workflow: Commit

List of Changes

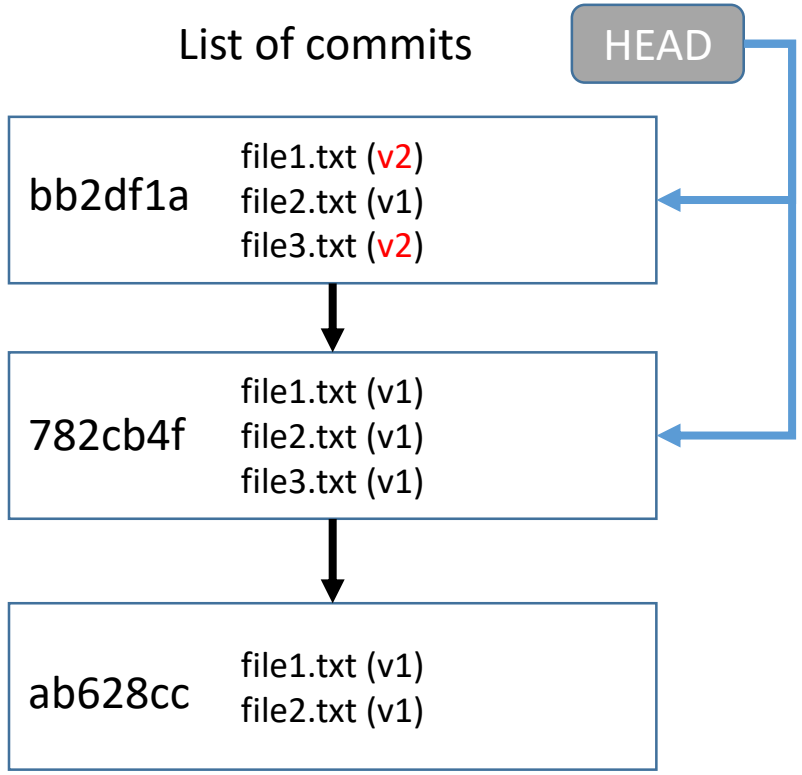
Staging Area

In file1.txt: add the line “here is a new line!” between lines 3 and 4

In file3.txt: delete line 27

Commit the currently staged differences

```
git commit -m "fixed bug in file1 and file3"
```



git status

Shows files differing between the staging area and the working directory (i.e. unstaged changes), the staging area and HEAD (i.e. changes ready to commit), and untracked files

```
Aaron@HELIOS ~/Dropbox/Dropbox Documents/98174/testing (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   demo.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   demo.txt
        modified:   one.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        new_file.txt
```

git diff

Example use:

(show unstaged changes)

git diff

(show staged changes)

git diff --cached

- Shows unstaged changes or staged changes

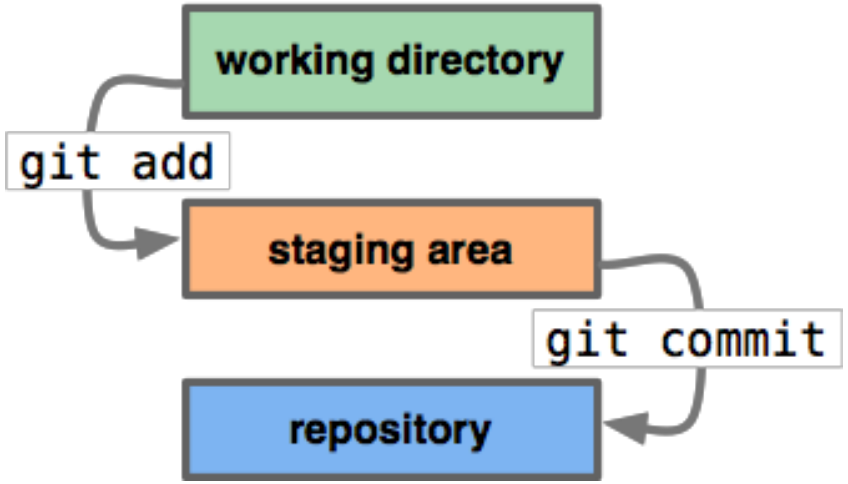
git show

Example use:

git show [commit hash (default is HEAD)]

- Shows the changes in the specified commit

git add



Example use:

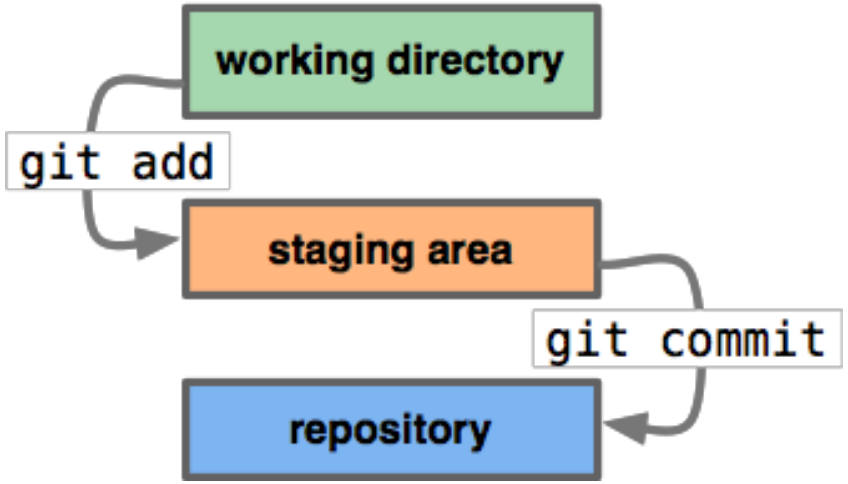
git add file1.txt file2.txt

(or)

git add . (adds changes to all files in directory)

- Creates a commit out of a snapshot of the staging area, and updates HEAD.

git commit



Example use:

git commit

(or)

git commit –m “commit message goes here”

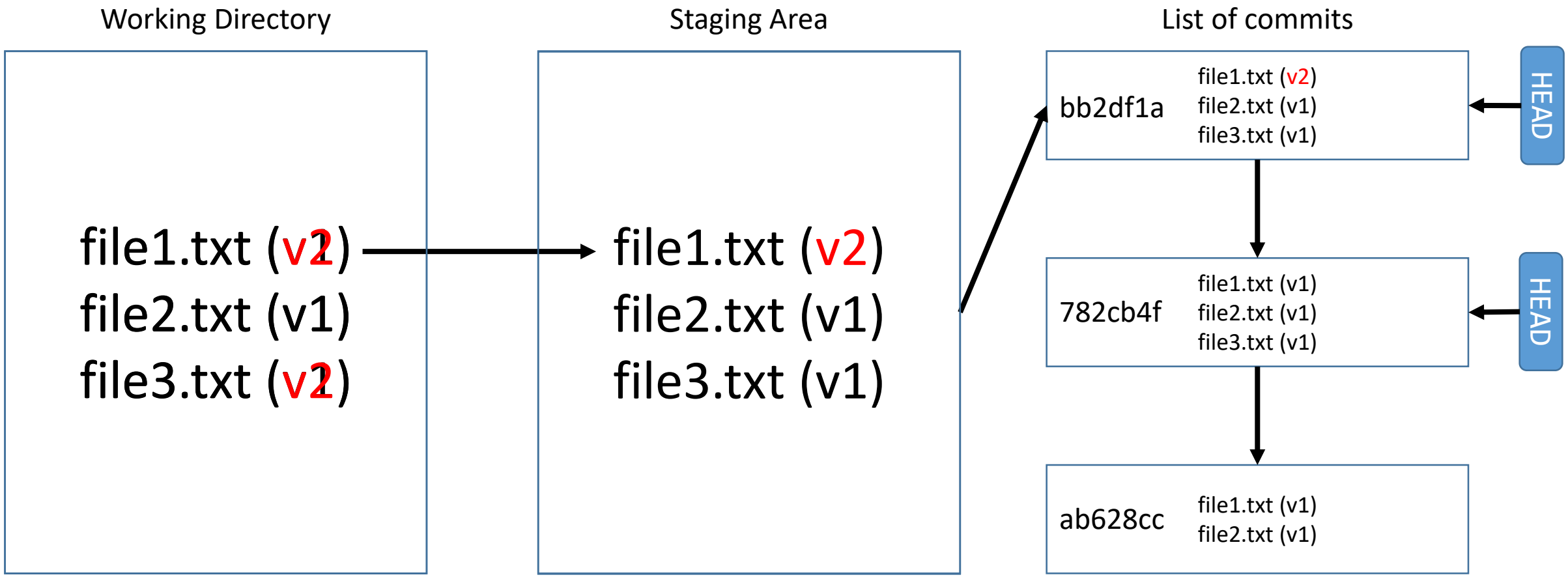
- Creates a commit out of a snapshot of the staging area, and updates HEAD.

Aside: commit HEAD

- The “most recent commit” has a special name: HEAD

```
* 250a199 - (HEAD -> master, origin/master, origin/HEAD) Build: Drop io.js testing
* d3d8d97 - Tests: Provide equal() arguments in correct order (actual, expected) (
* 0e98243 - Data: avoid using delete on DOM nodes (Tue Sep 8 14:22:54 2015) <Jason
* d4def22 - Manipulation: Switch rnoInnerhtml to a version more performant in IE (
* 1b566d3 - Tests: Really fix tests in IE 8 this time (Tue Sep 8 13:02:35 2015) <M
* 5914b10 - Tests: Make basic tests work in IE 8 (Tue Sep 8 12:43:08 2015) <Michał
```

Review: The Git Commit Workflow (Edit, Add, Commit)

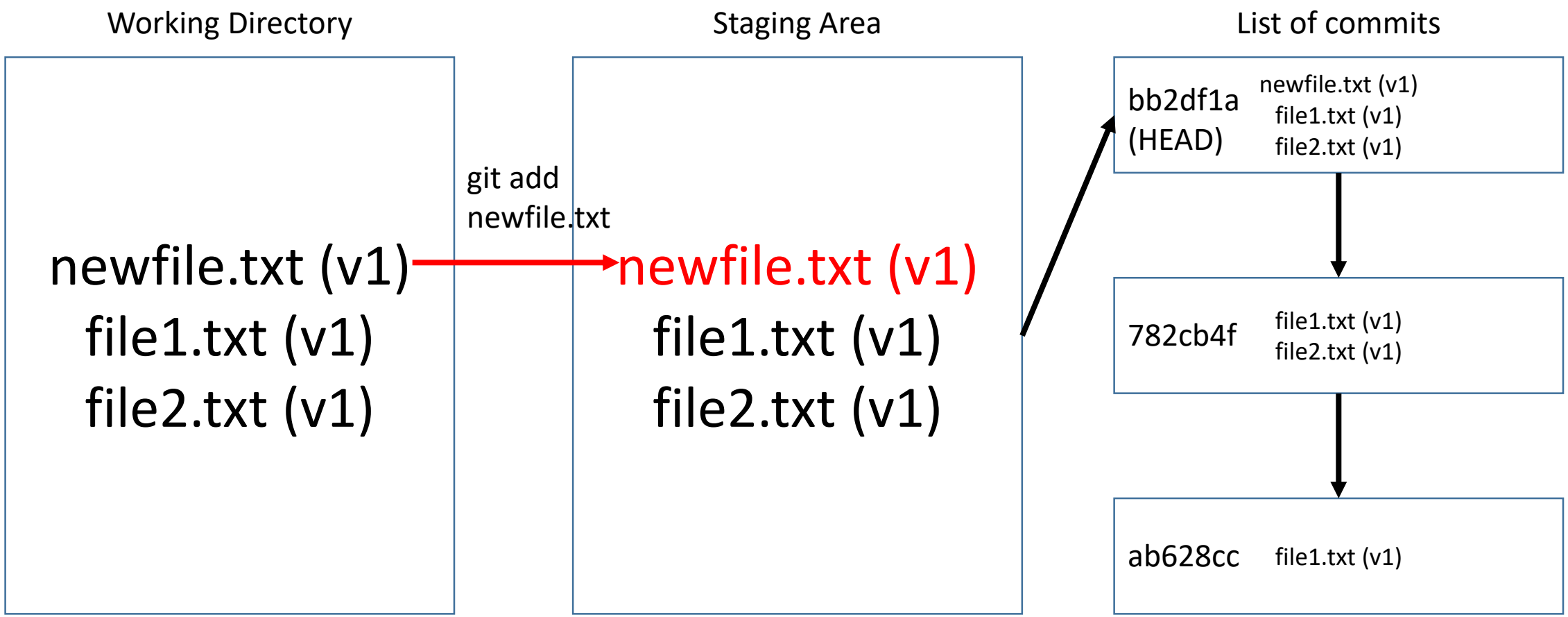


1. Make changes to files
`vim file1.txt file3.txt`

2. Add changes to the staging area
`git add file1.txt`

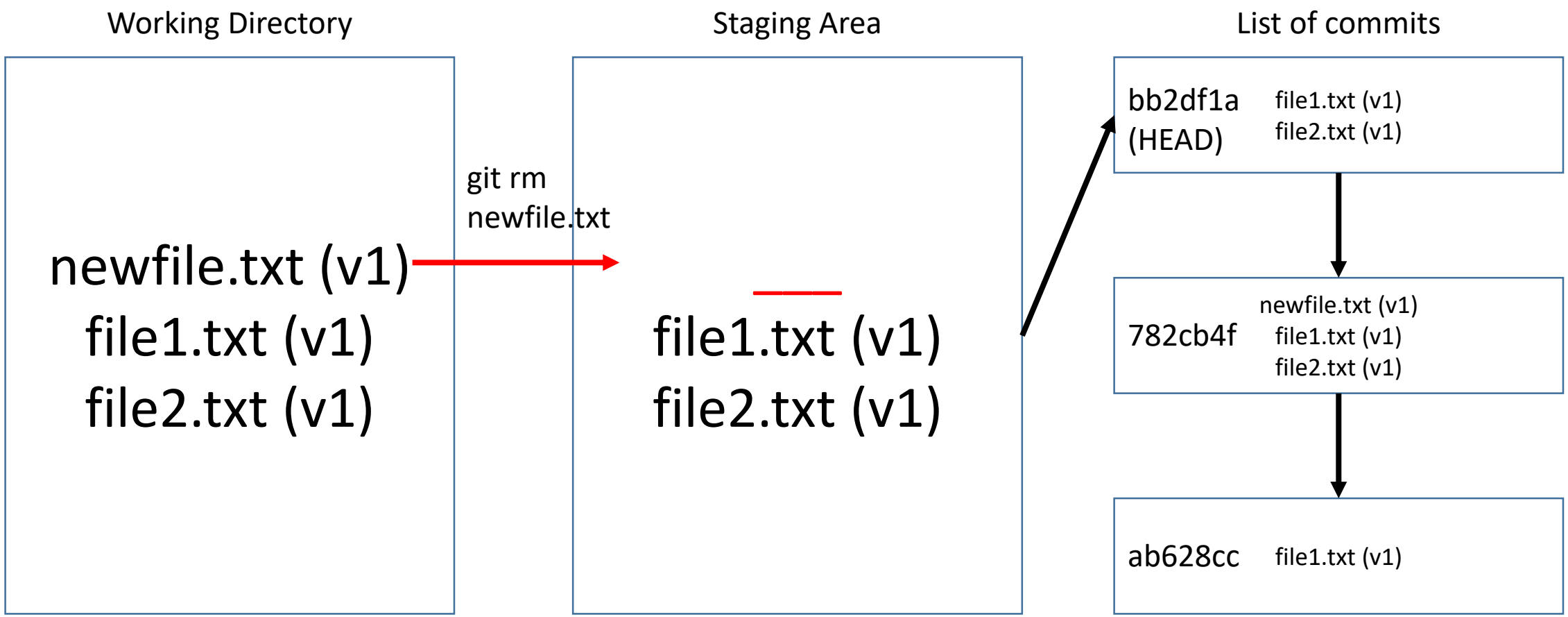
3. Commit changes in staging area
`git commit -m “fixed bug in file1.txt”`

What about new files?



No difference from an edit, use git add newfile.txt.

What about removing files?



git rm newfile.txt (also deletes newfile.txt from working directory!)

What if I want to undo changes in the Working Dir?

Working Directory

~~coolfile.txt (v2)~~
coolfile.txt (v1)
file1.txt (v1)
file2.txt (v1)

Staging Area

List of commits

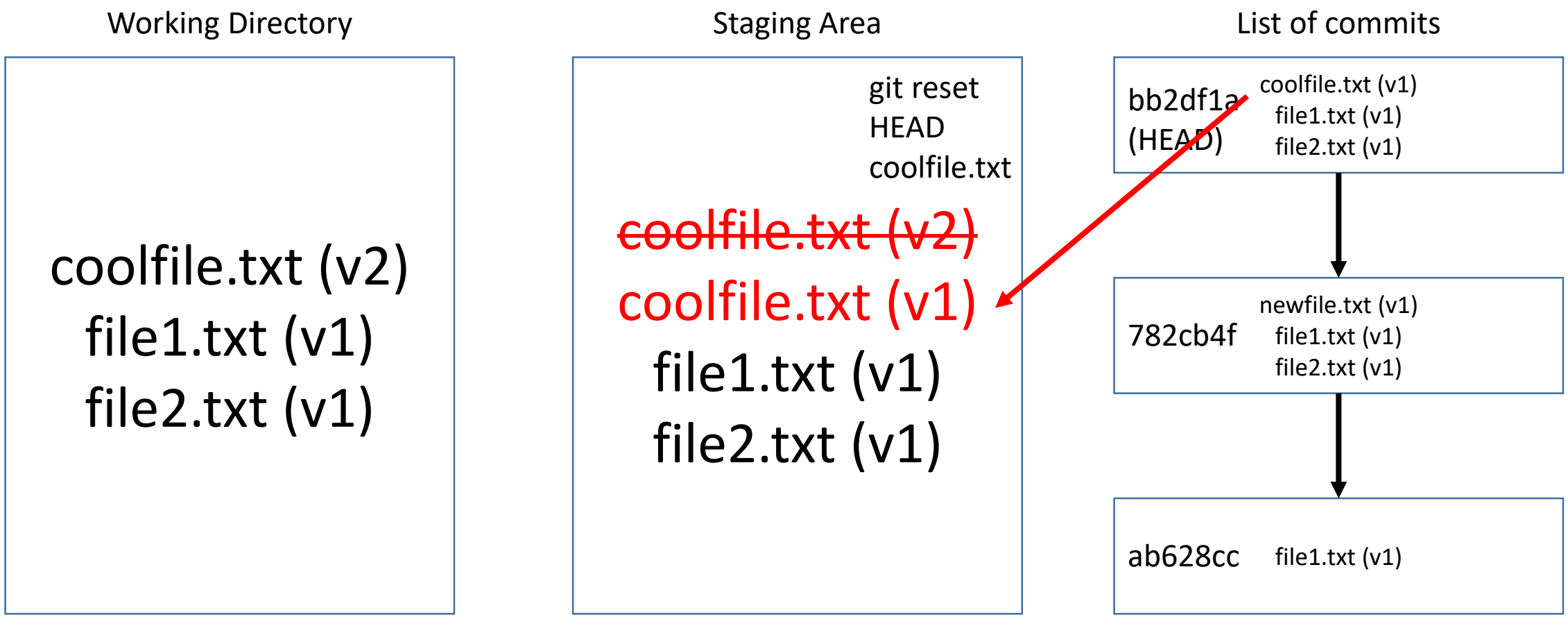
bb2df1a (HEAD)
coolfile.txt (v1)
file1.txt (v1)
file2.txt (v1)

782cb4f
newfile.txt (v1)
file1.txt (v1)
file2.txt (v1)

ab628cc
file1.txt (v1)

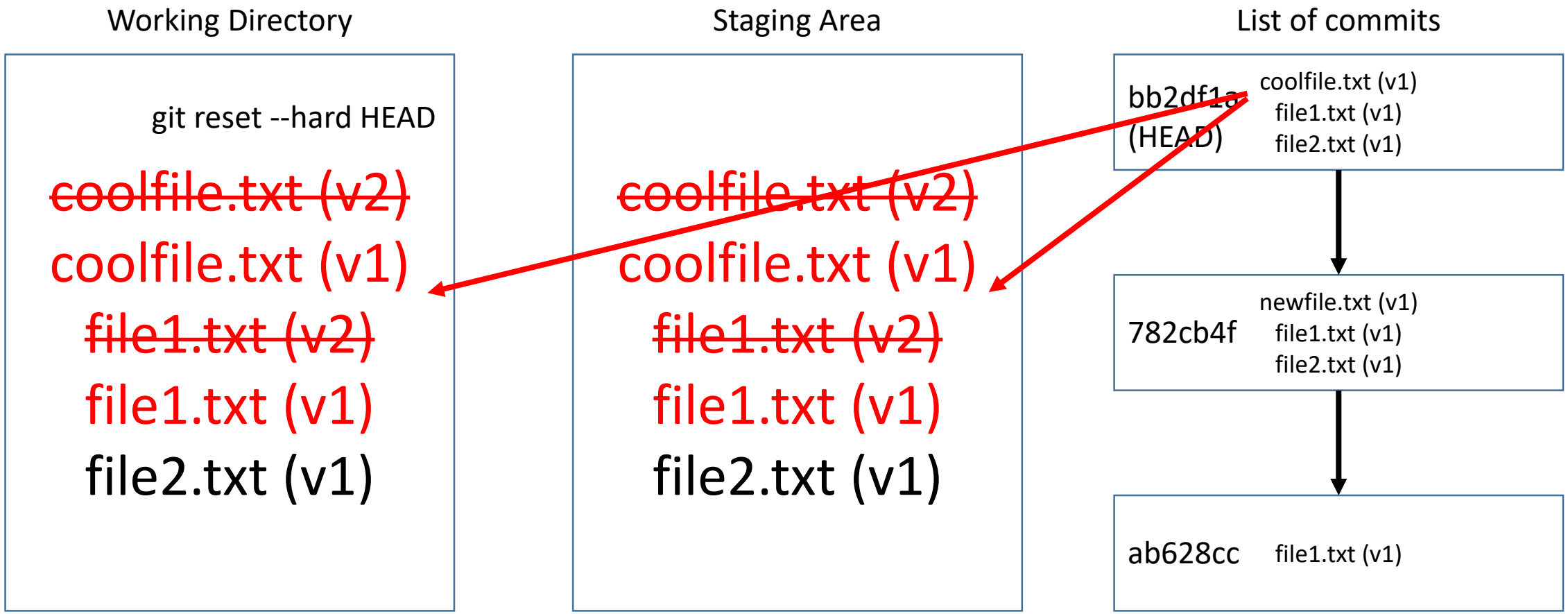
git checkout -- coolfile.txt (Note staging area is unaffected)

What if I want to ‘unstage’ a file?



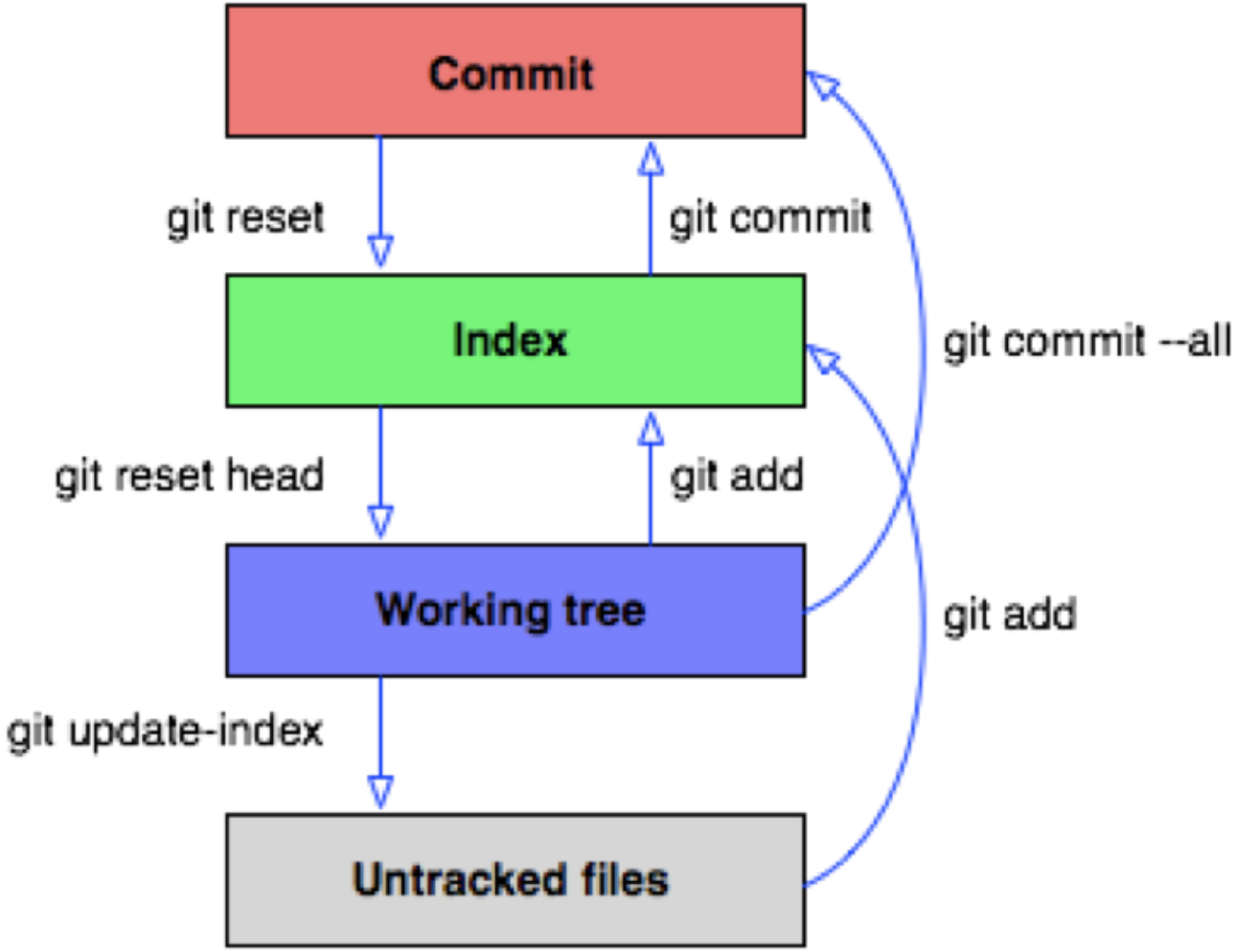
git reset HEAD coolfile.txt (Note WD is unaffected)

What if I want to start over and go back to exactly what the HEAD looks like (in both WD and SA)?

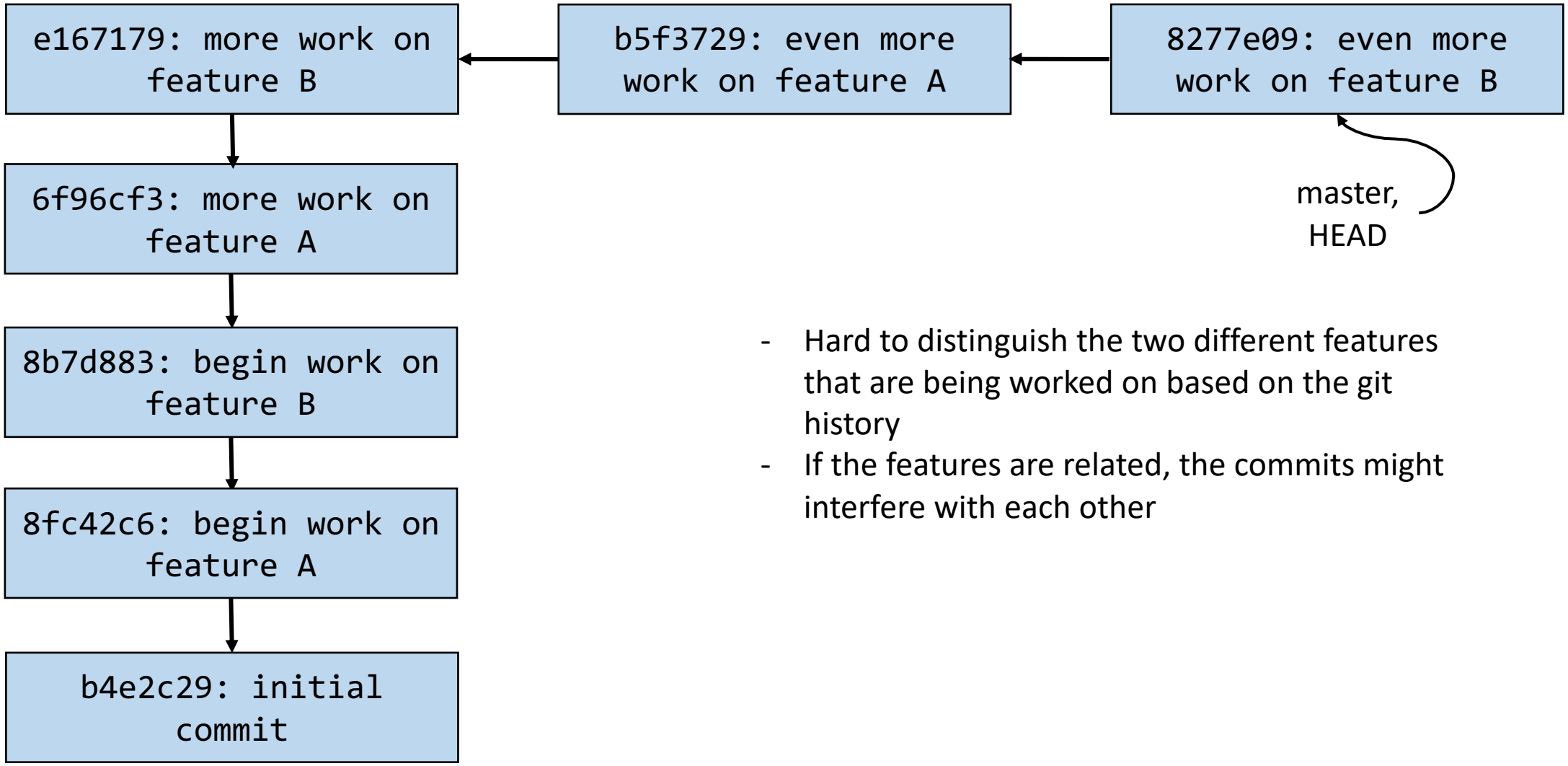


git reset --hard HEAD (overwrites entire WD!)

Last Time

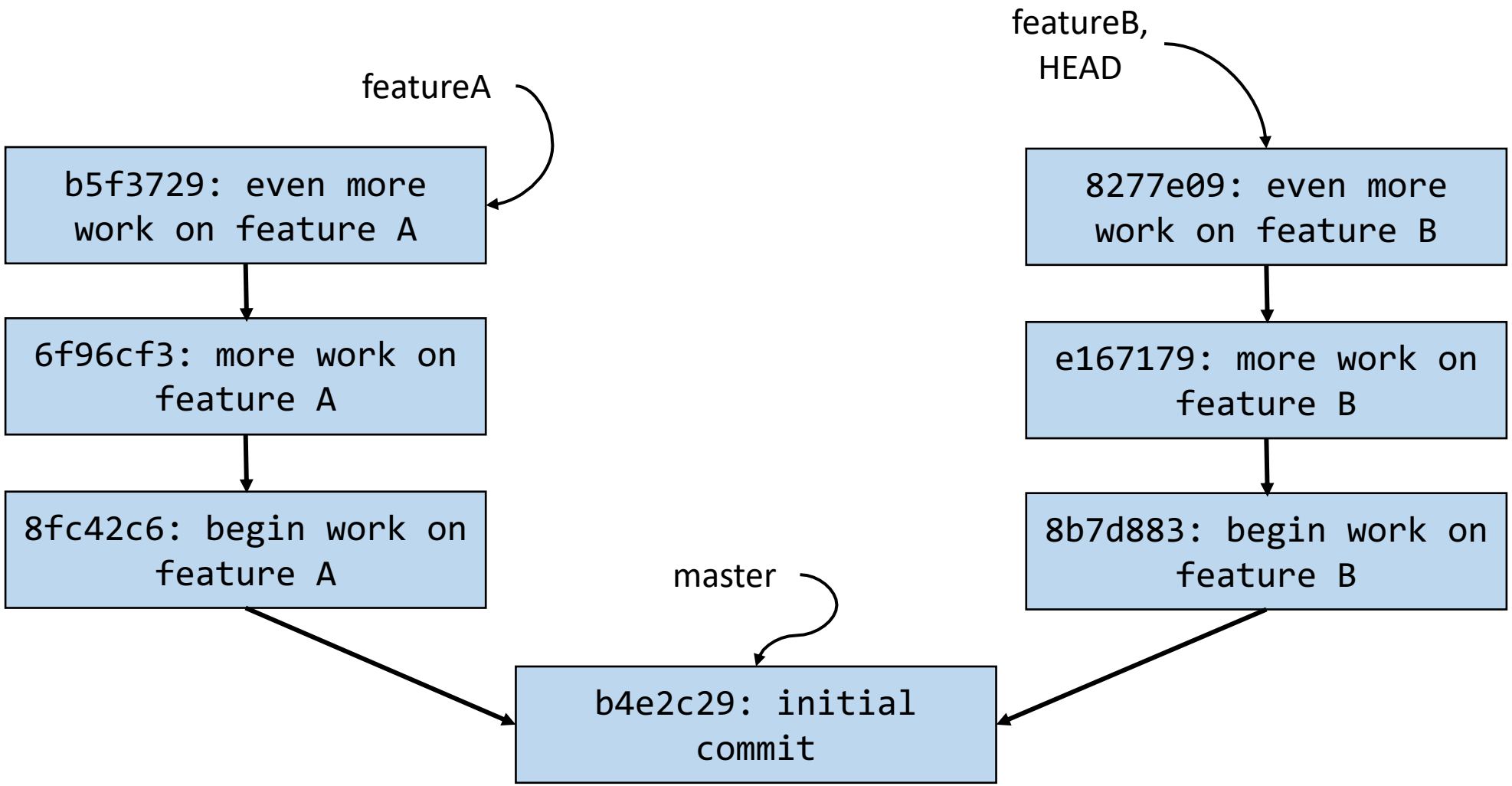

Commit
git reset
git commit
Index
git commit --all
git reset head
git add
Working tree
git add
git update-index
Untracked files

Scenario: You work on two features at once in a project



- Hard to distinguish the two different features that are being worked on based on the git history
- If the features are related, the commits might interfere with each other

Solution: Non-linear development via branches



git branch

Example use:

```
git branch
```

```
Aaron@HELIOS ~/Dropbox
$ git branch
  feature-2
* master
  new-feature
```

- Lists all the local branches in the current repository and marks which branch you're currently on
  - Where are "you"? Well, you're always at HEAD. Usually, you're also at a branch as well.
- The default branch in a repository is called "master"

`git branch <newbranchname>`

Example use:

```
git branch develop
```

- Creates a new branch called “develop” that **points** to wherever you are right now (i.e. wherever HEAD is right now)

`git checkout <branchname>`

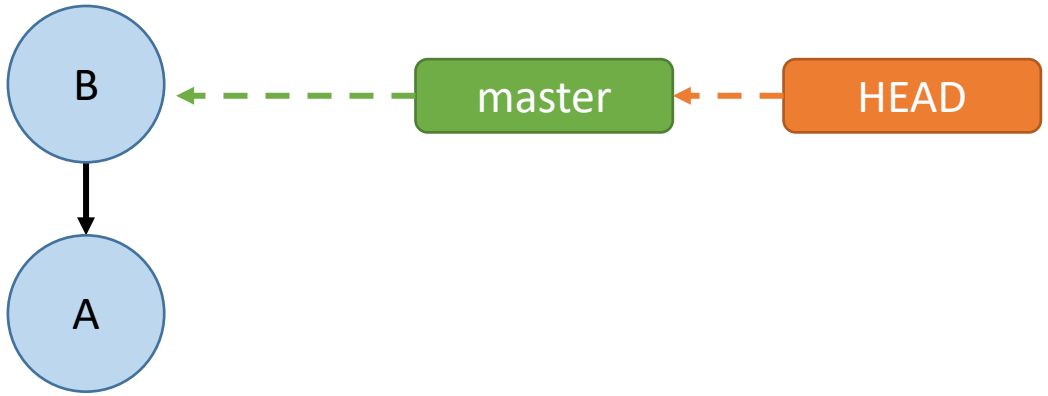
Example use:

```
git checkout develop
```

- Switches to the branch named “develop”
- Instead of a branch name, you can also put a commit hash
  - More on this next lecture

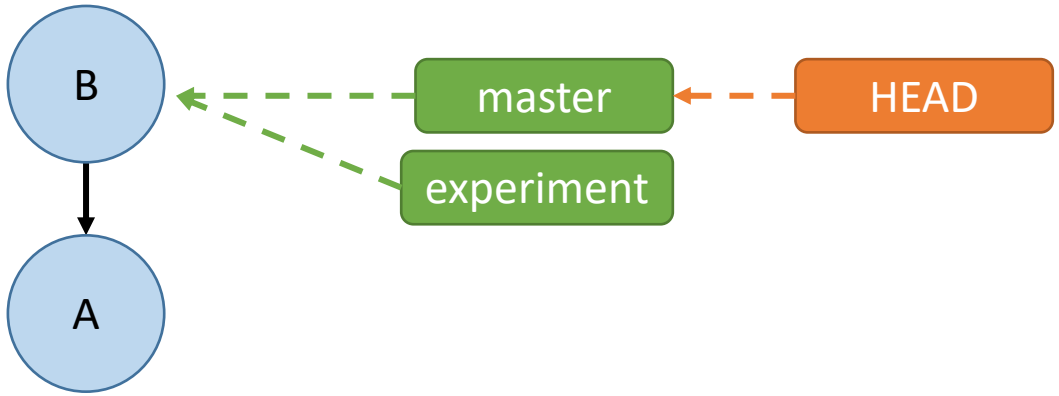
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment



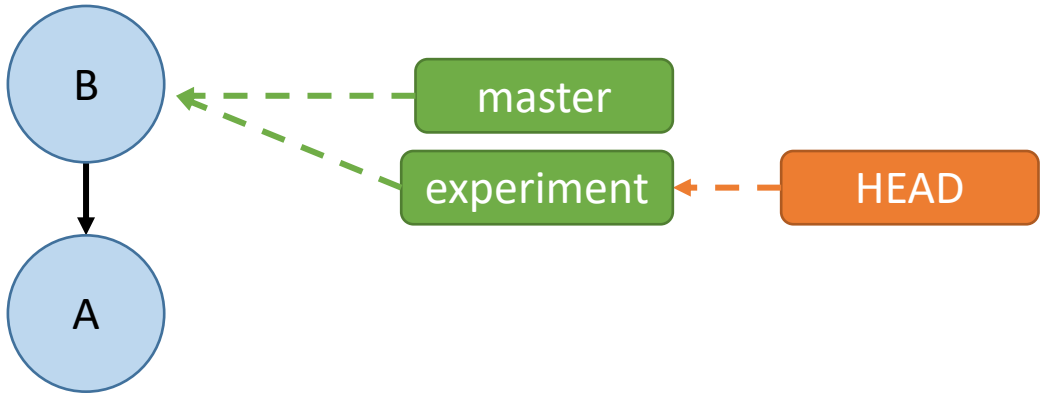
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment



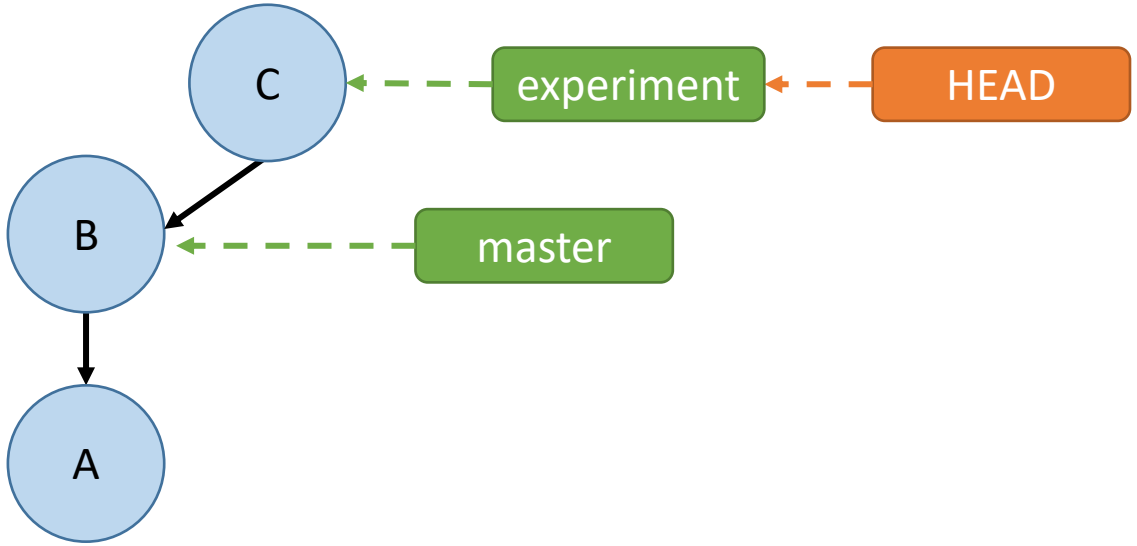
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment
5. git commit –m "C"



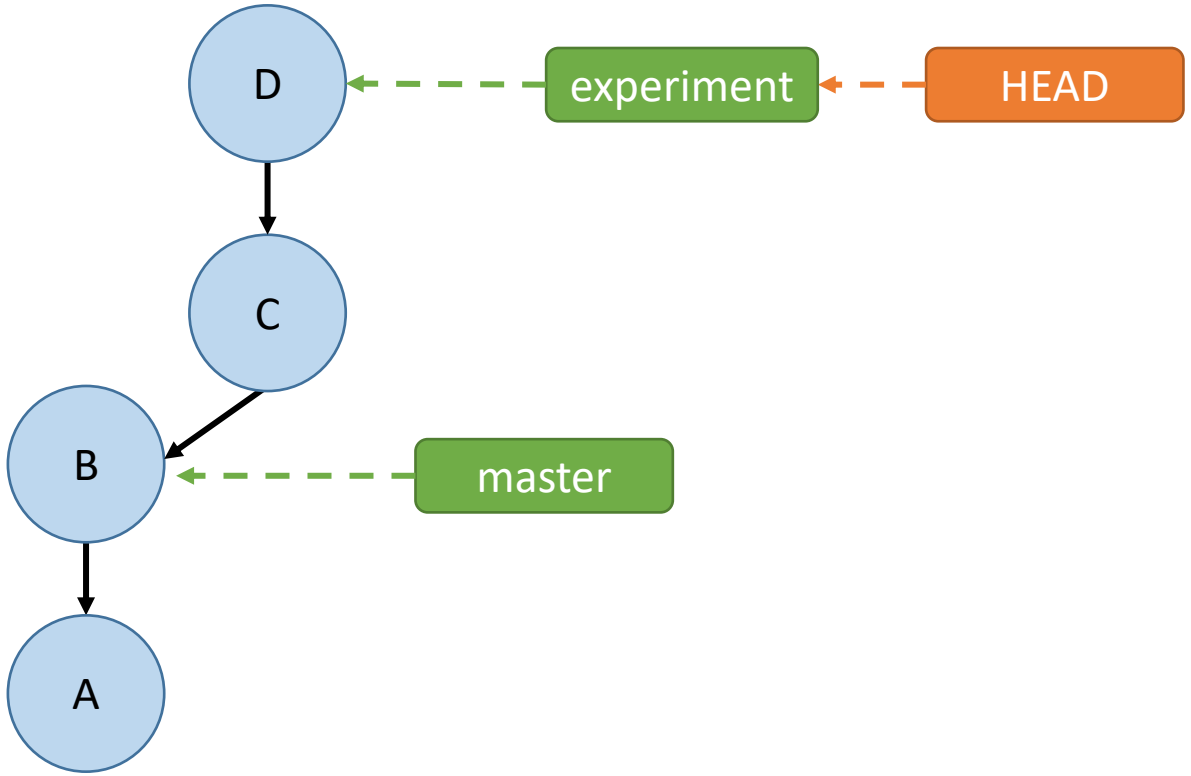
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment
5. git commit –m "C"
6. git commit –m "D"



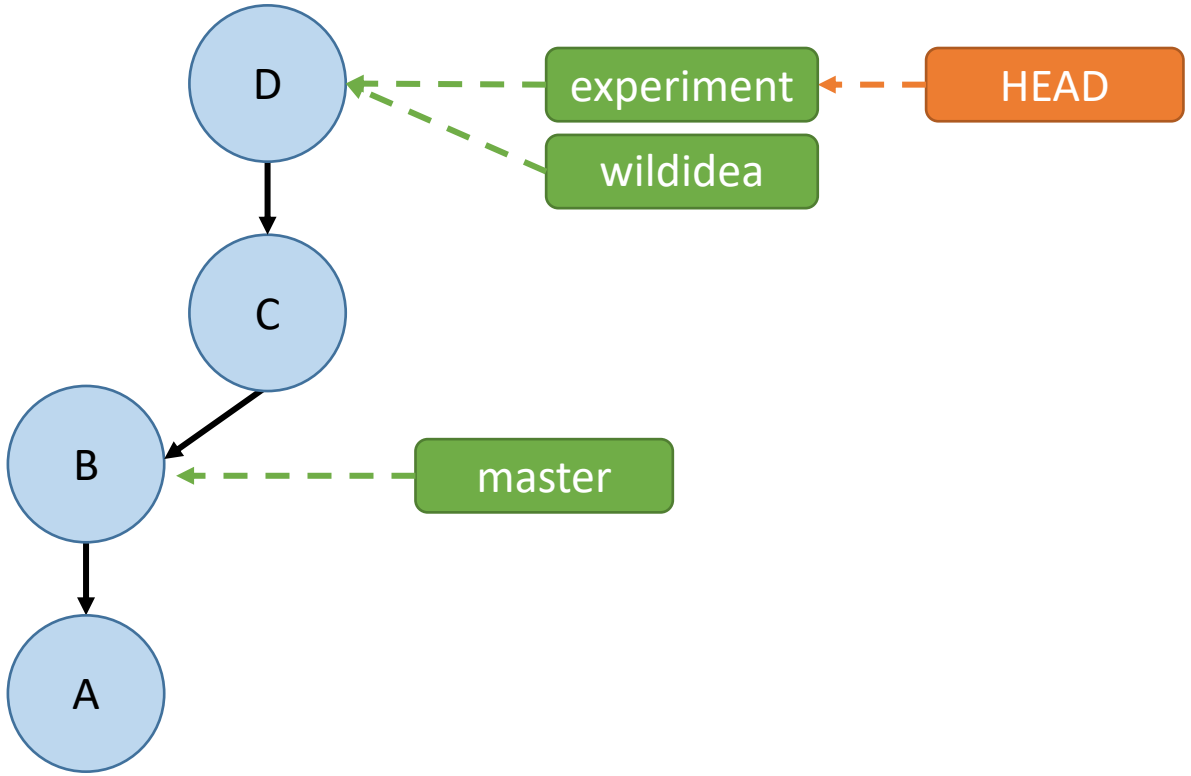
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment
5. git commit –m "C"
6. git commit –m "D"
7. git branch wildidea



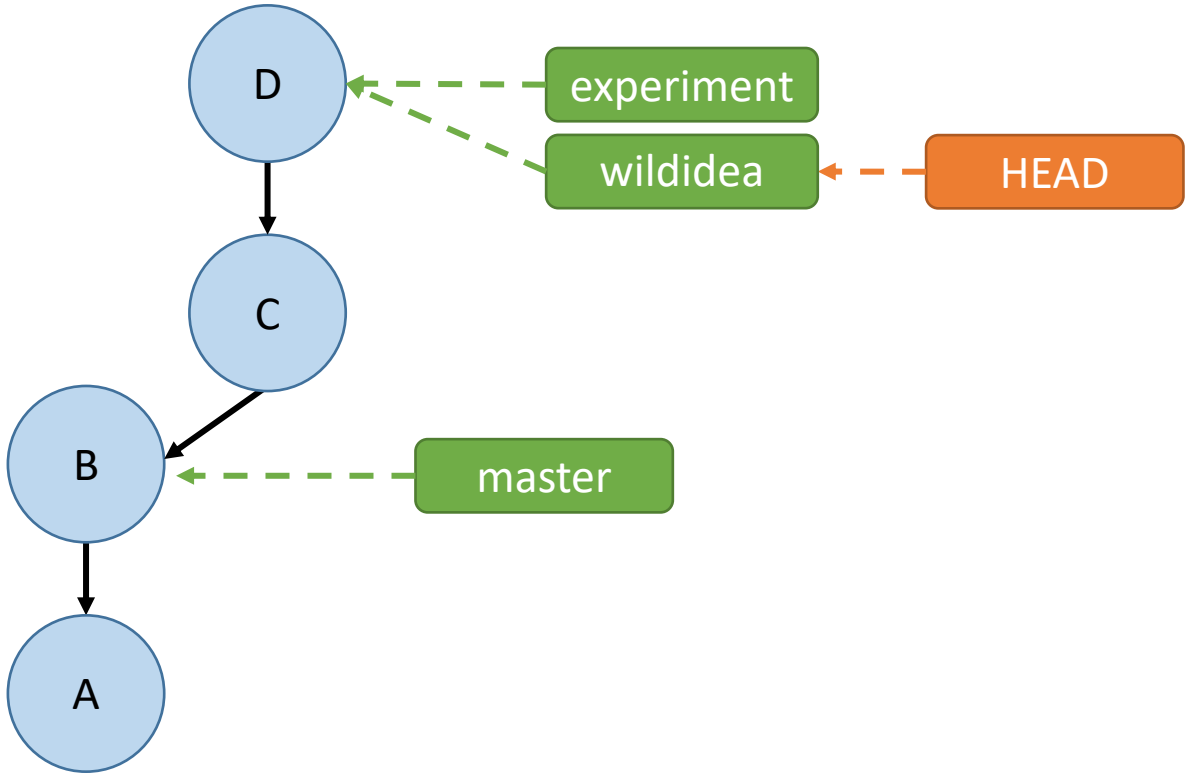
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment
5. git commit –m "C"
6. git commit –m "D"
7. git branch wildidea
8. git checkout wildidea



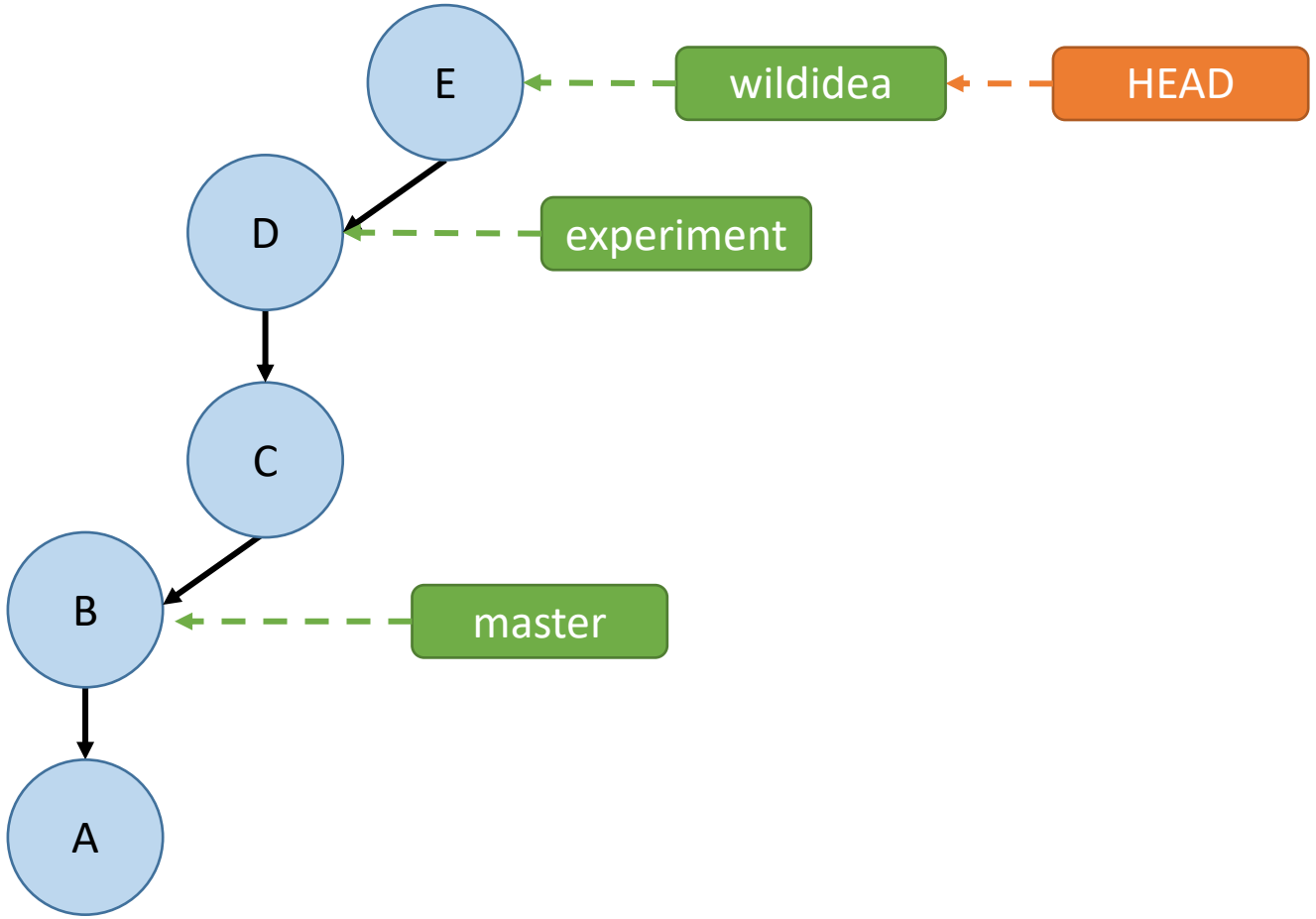
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment
5. git commit –m "C"
6. git commit –m "D"
7. git branch wildidea
8. git checkout wildidea
9. git commit –m "E"



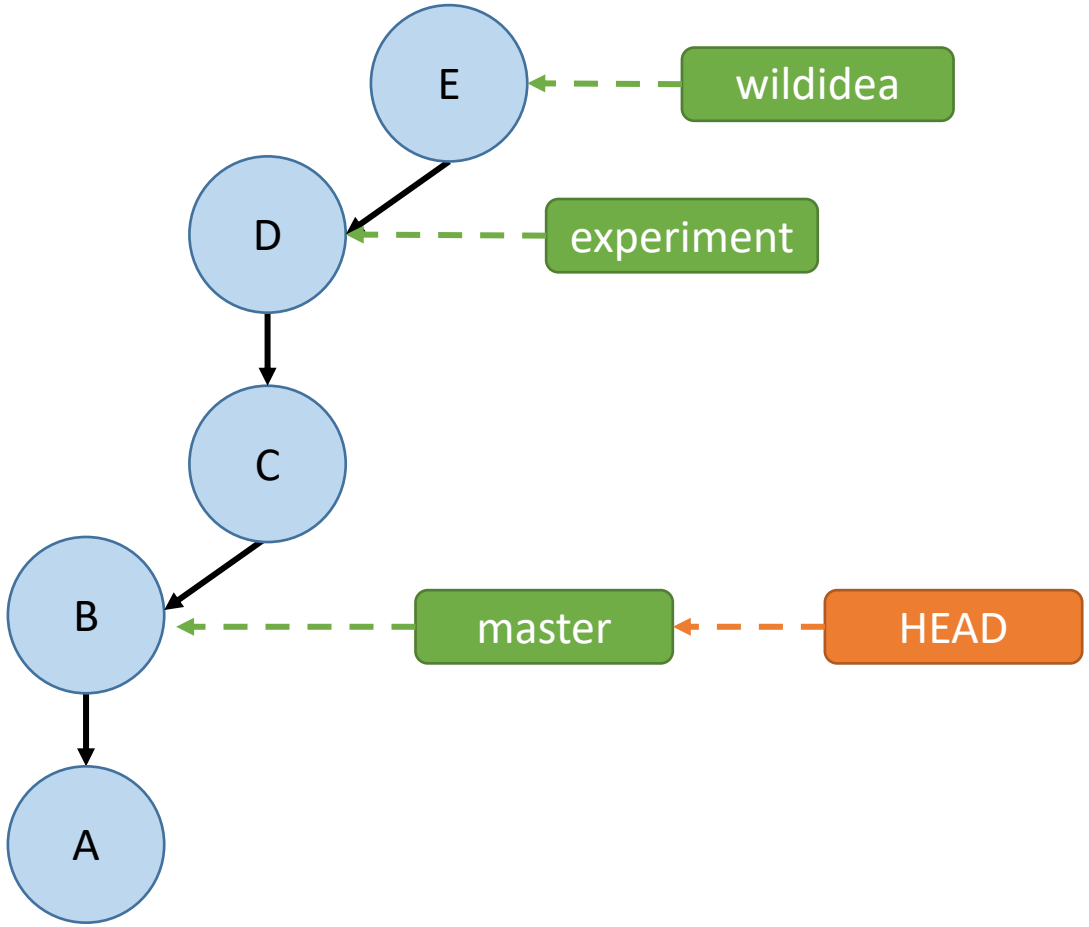
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment
5. git commit –m "C"
6. git commit –m "D"
7. git branch wildidea
8. git checkout wildidea
9. git commit –m "E"
10. git checkout master



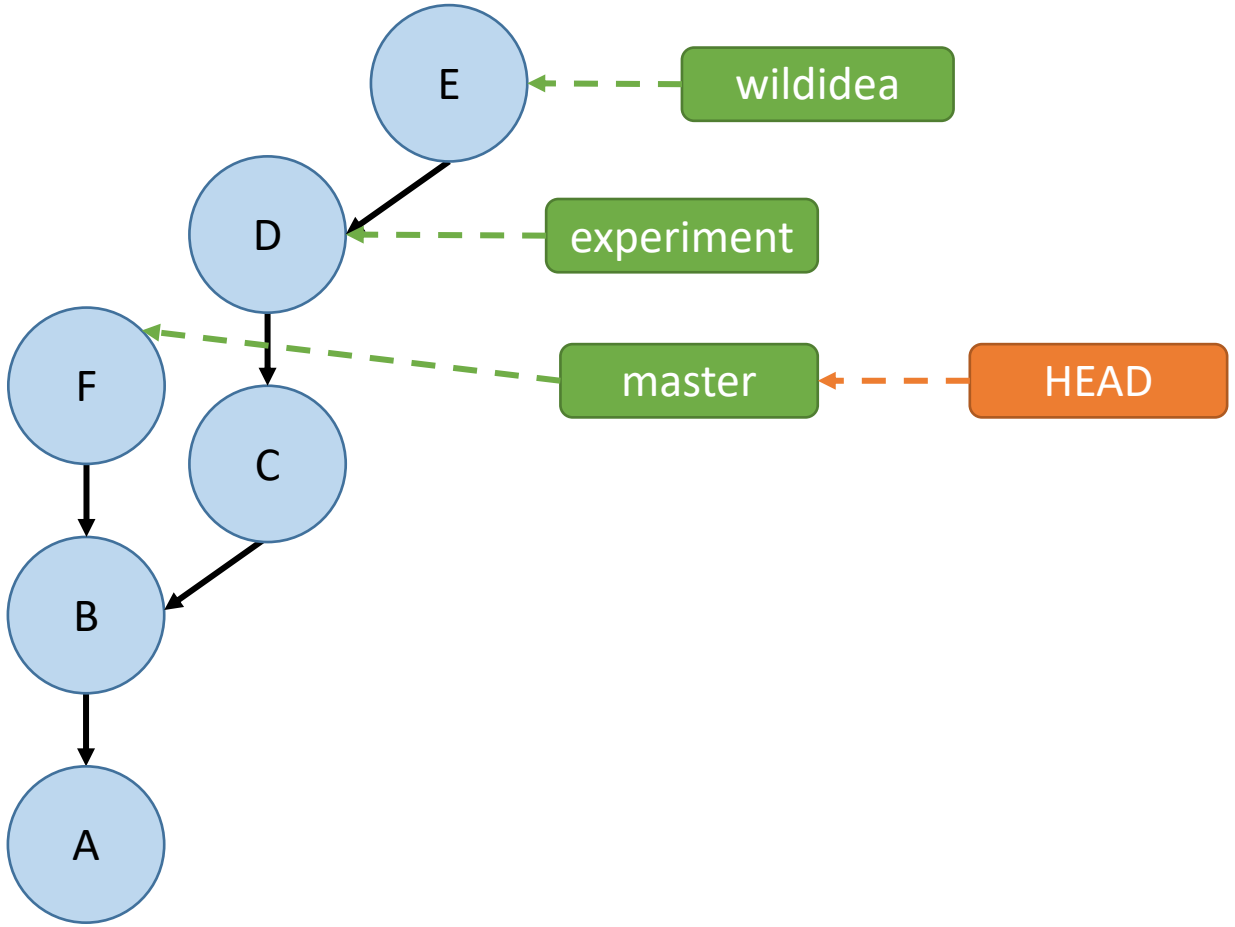
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment
5. git commit –m "C"
6. git commit –m "D"
7. git branch wildidea
8. git checkout wildidea
9. git commit –m "E"
10. git checkout master
11. git commit –m "F"

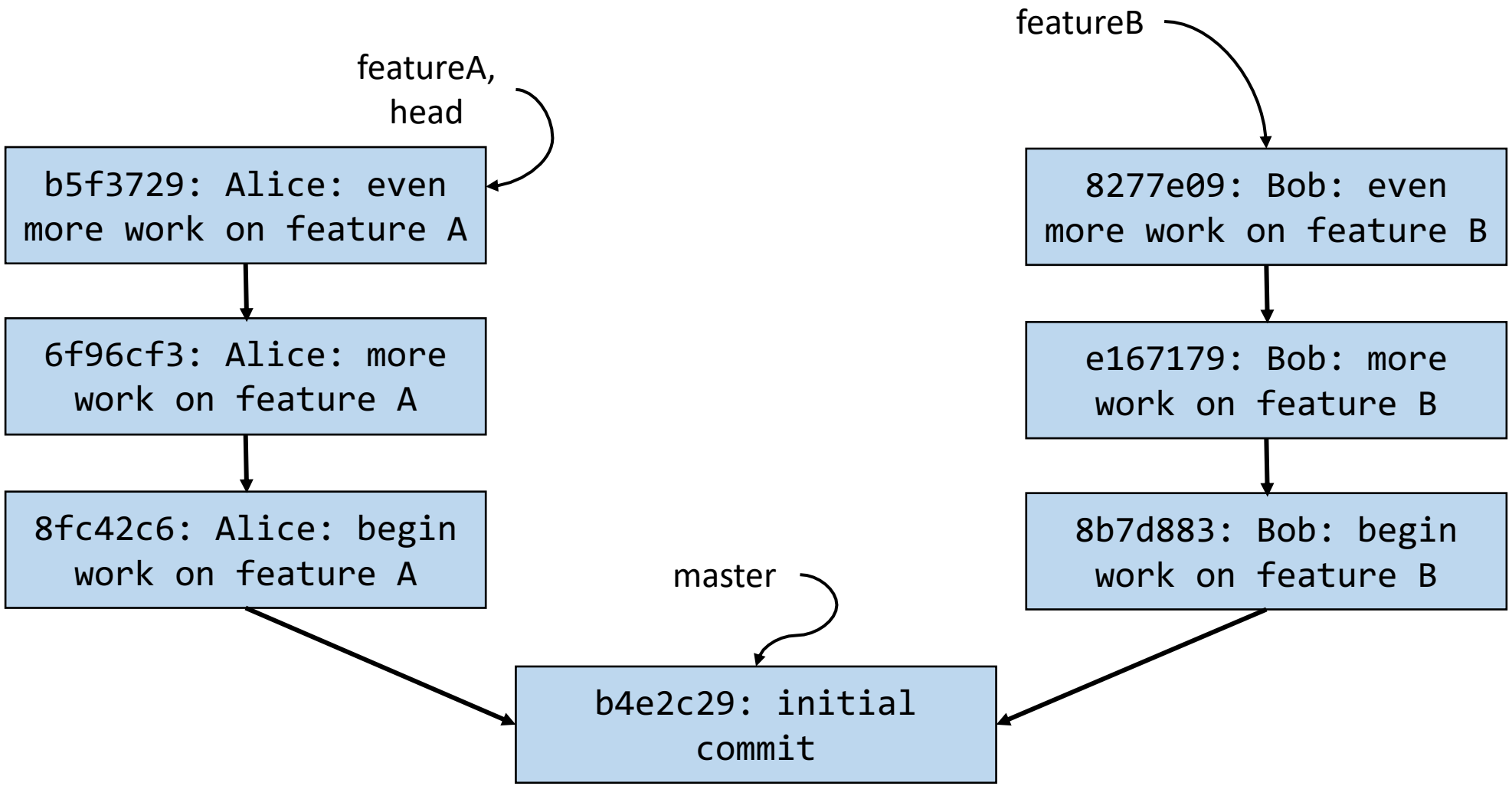


Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment
5. git commit –m "C"
6. git commit –m "D"
7. git branch wildidea
8. git checkout wildidea
9. git commit –m "E"
10. git checkout master
11. git commit –m "F"

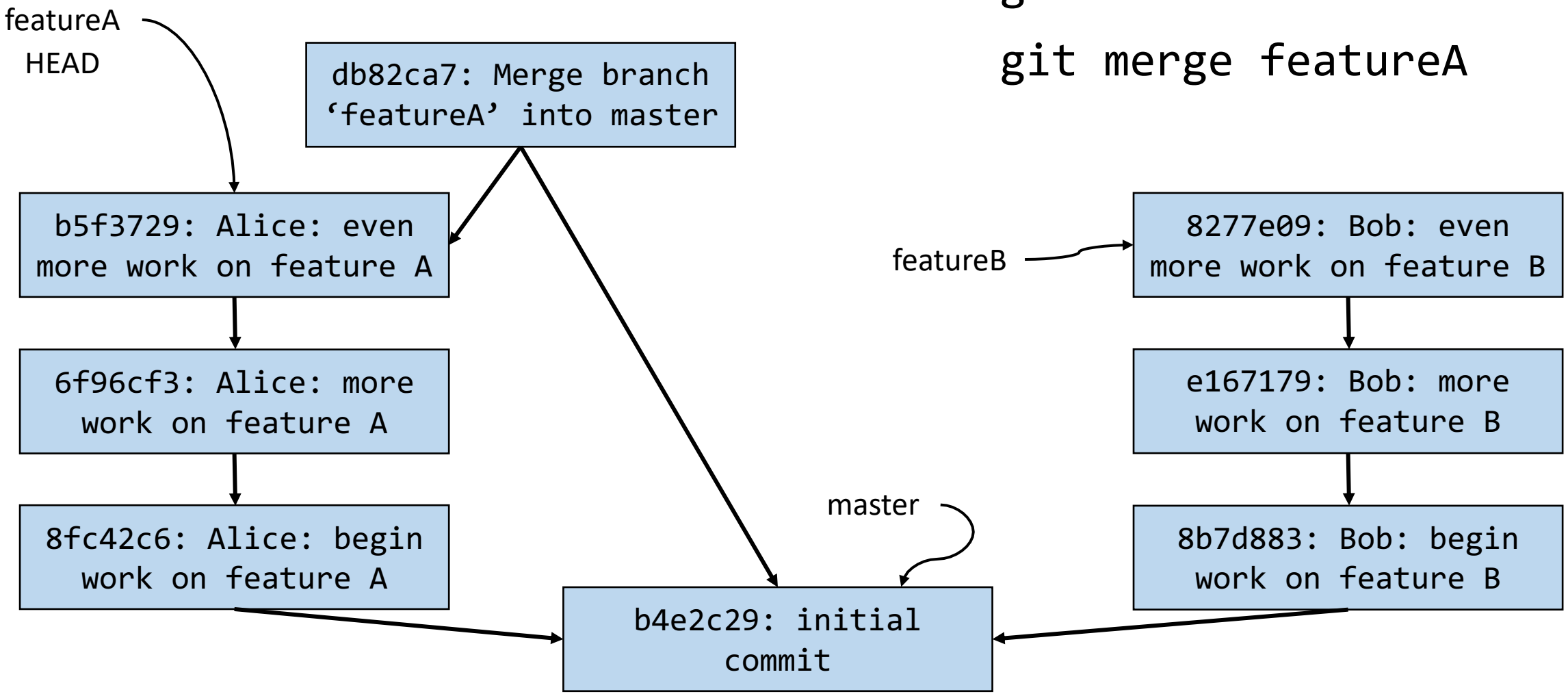


How do we bring branches back together?



How do we bring branches back together?

```
git checkout master
git merge featureA
```



`git merge <branch_to_merge_in>`

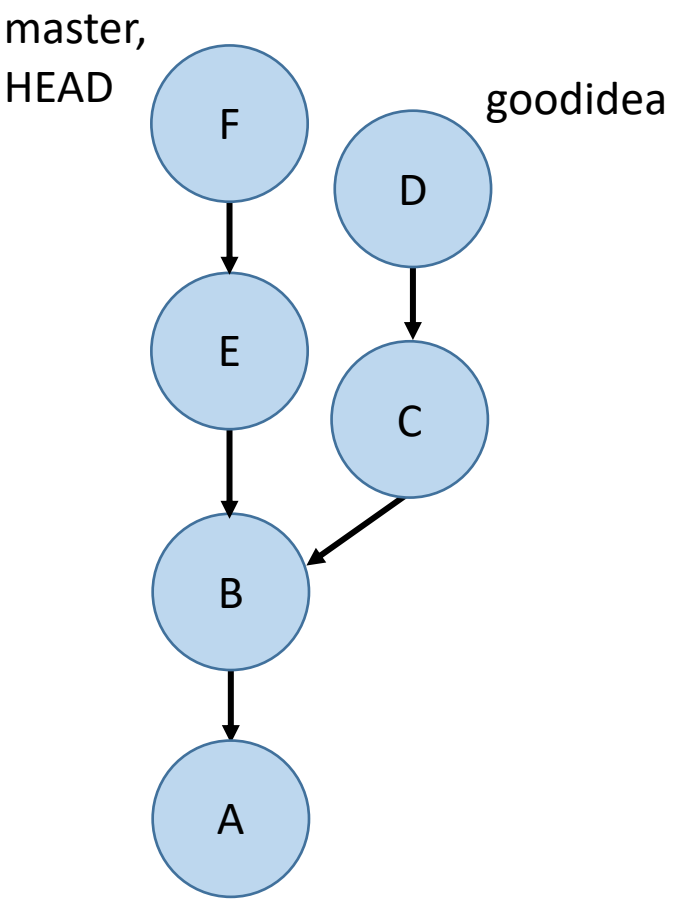
Example use:

```
git merge featureA
```

- Makes a new merge commit on the CURRENT branch that brings in changes from featureA

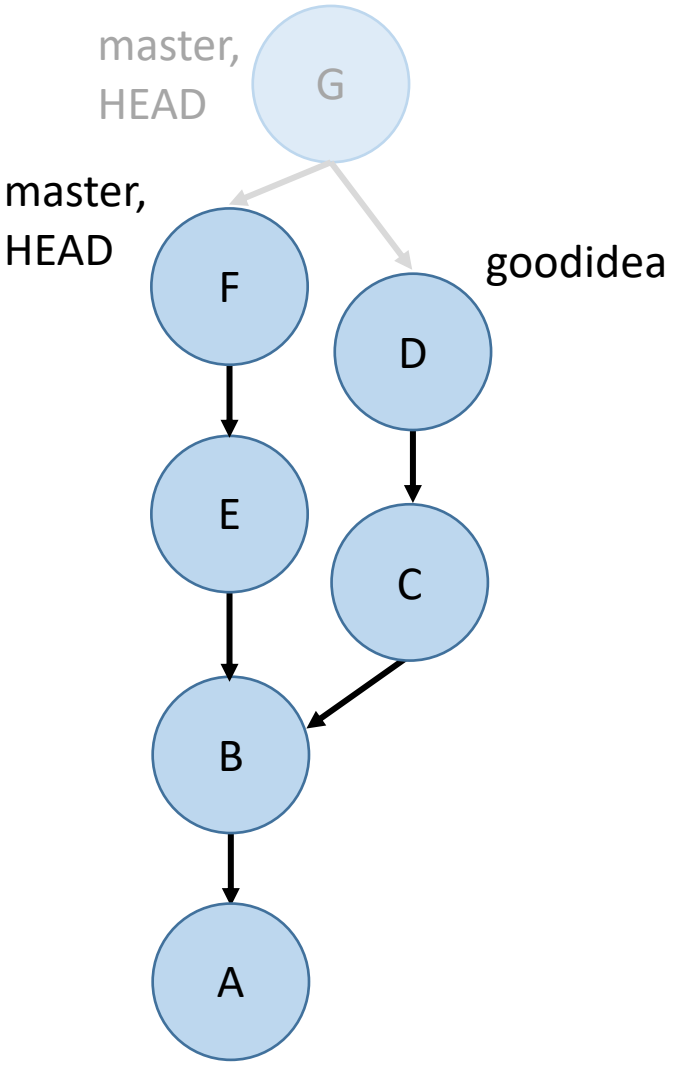
Most cases: Merging with possible conflicts

- Let's say I'm on master (as denoted by HEAD) and I want to merge goodidea into master.
- git merge goodidea



Most cases: Merging with possible conflicts

- Let's say I'm on master (as denoted by HEAD) and I want to merge goodidea into master.
- git merge goodidea
- At this point, if bringing in all the changes from goodidea do not conflict with the files in master, then a new commit is created (you'll have to specify a commit message) and we're done.
- Otherwise…git just goes halfway and stops.



MERGE CONFLICT

```
:( andrew@hydreigon ~/temp3
03:57 PM (master)$ git merge goodidea
Auto-merging D
CONFLICT (add/add): Merge conflict in D
Automatic merge failed; fix conflicts and then commit the result.
```

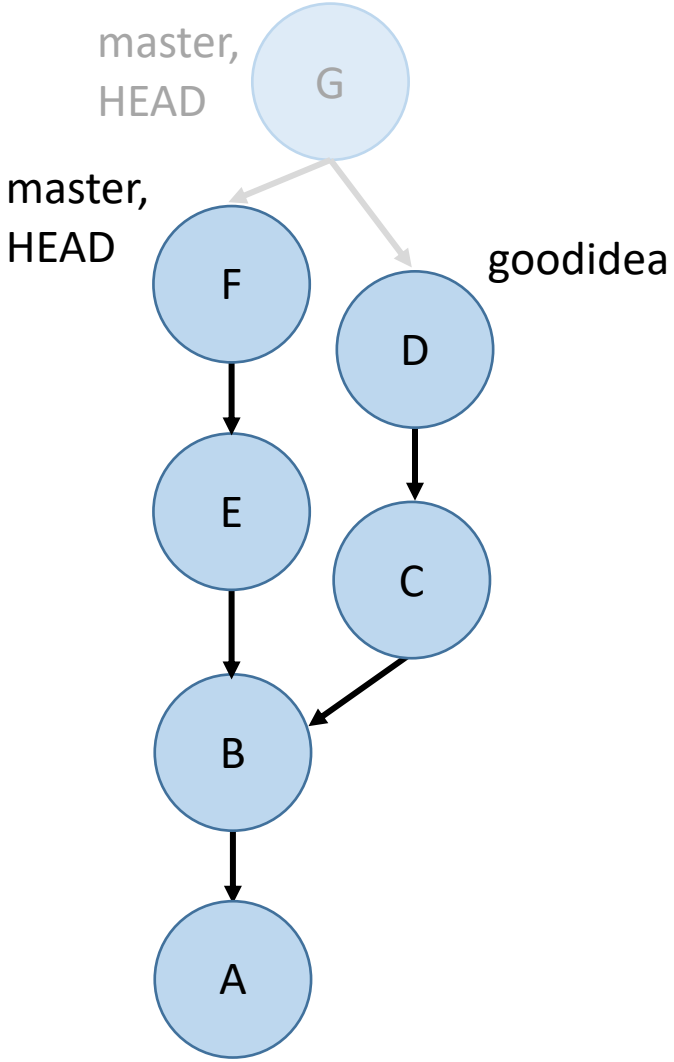
```
:( andrew@hydreigon ~/temp3
03:57 PM (master)$ git s
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Changes to be committed:

        new file:   C

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both added:      D
```



MERGE CONFLICT

```
This file is demo.txt

<<<<<<< HEAD
Here is another line. modified in master
=======
Here is another line. modified in goodidea
>>>>>>> goodidea
```

“How to fix a merge conflict”

- Run `git status` to find the files that are in conflict.
- For each of these files, look for lines like “<<<<<< HEAD” or “>>>>>> 3de67ca” that indicate a conflict.
- Edit the lines to match what you want them to be.
- After you finish doing this for each conflict in each file, `git add` these conflicted files and run `git commit` to complete the merge.

```
:( andrew@hydreigon ~/temp3
03:57 PM (master)$ git s
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Changes to be committed:

        new file:   C

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both added:      D
```

Scenario: you want to switch branches, but you have uncommitted changes

```
:( Andrew@Gengar ~/temp
04:08 PM (coolfeature)$ git checkout master
error: Your local changes to the following files would be overwritten by checkout:
        A
Please, commit your changes or stash them before you can switch branches.
Aborting
```

What if you don’t want to commit?

git stash

Example use:

git stash

- Makes a “pseudo-commit” and puts it on a **stack** of stashed pseudo-commit.
- Message for stash is “WIP on <*branchname*>…”
- Use git stash save <message> to store stashes with better messages

git stash pop

Example use:

```
git stash pop
```

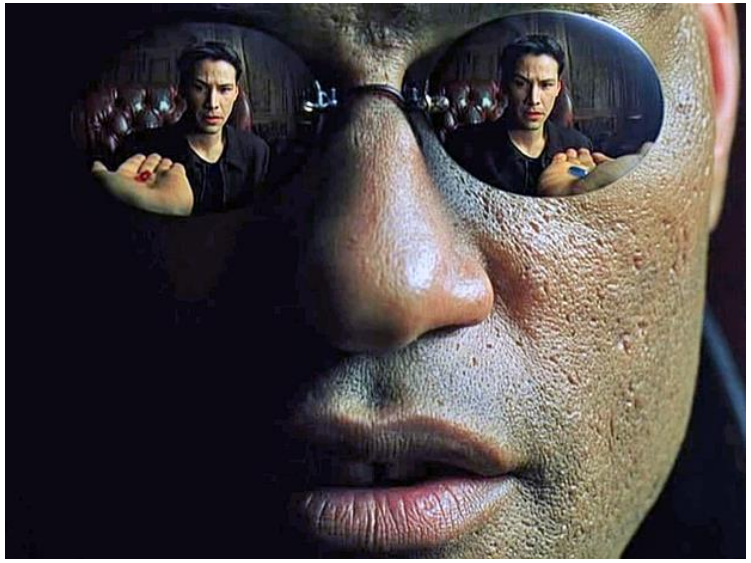
- Reapplies the top stashed change and removes it from the stash stack.

git stash show (-p) (stash@{<depth>})

Example use:

git stash show stash@{2}

- Show details about the stashed change at the specified depth, if given.



git stash apply (stash@{<depth>})

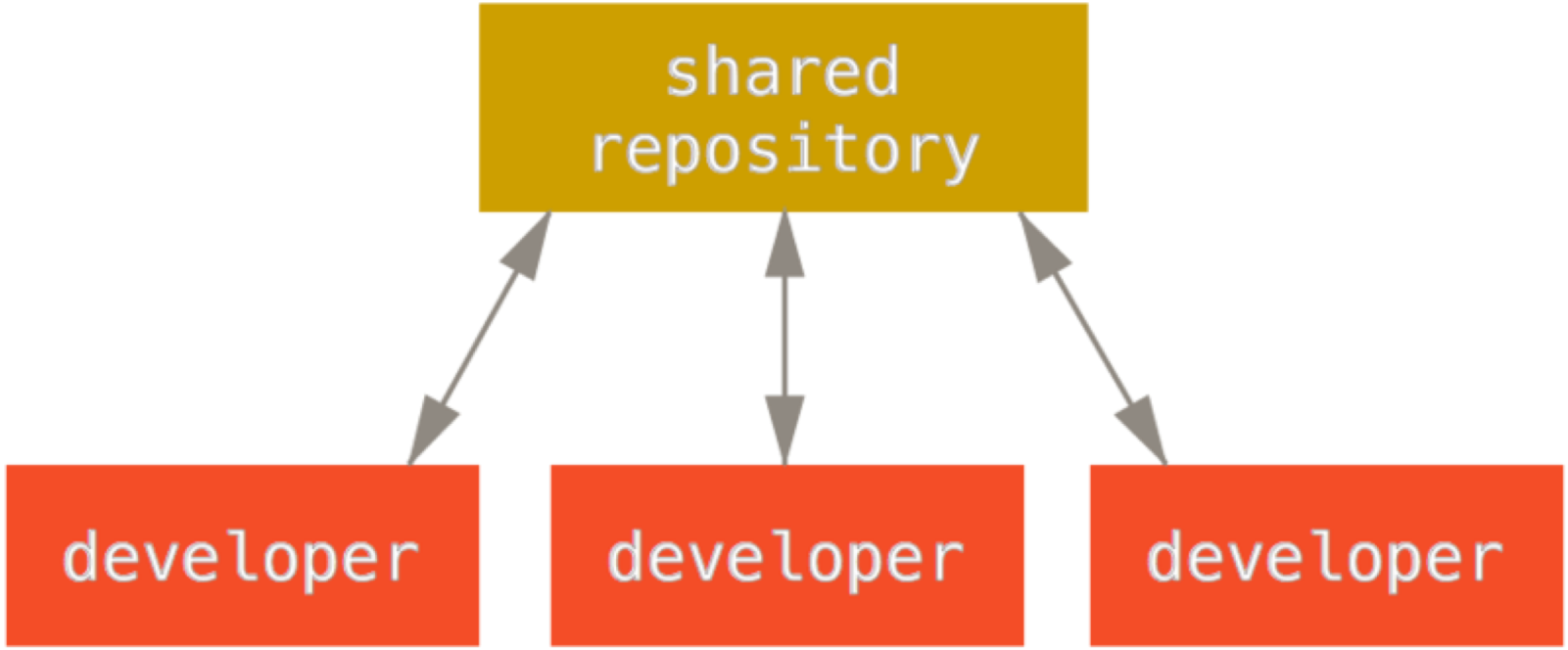
Example use:

git stash apply stash@{2}

- Reapplies the stashed change at the specified depth, if given.
- Depth is just another way of choosing from a list of saved stashes

Centralized Git Workflow

WHAT?! I thought Git was a **Distributed** Version Control System!



Pushing

```
$ git push origin master
```

Pushes the local branch called `master` to the branch called `master` on the remote named `origin`

This is how we move where remote branches **point to**

```
$ git remote -v
origin  https://github.com/aperley/dino-story.git (fetch)
origin  https://github.com/aperley/dino-story.git (push)
```