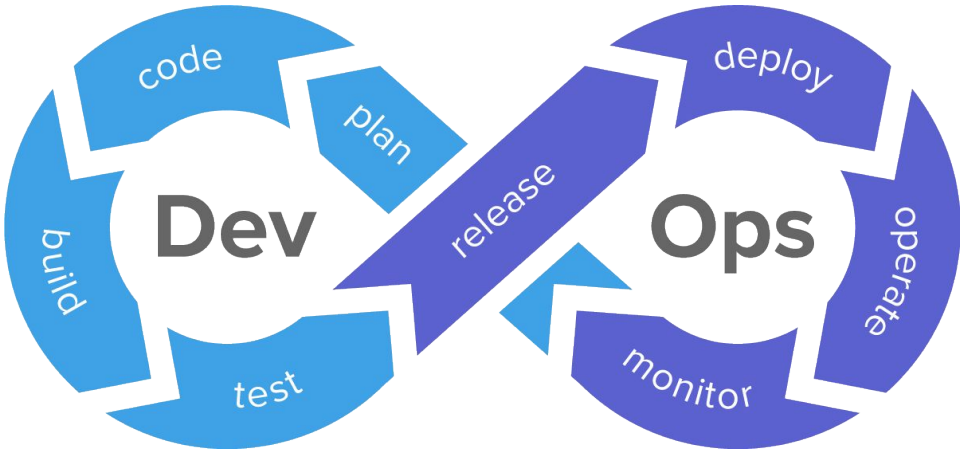
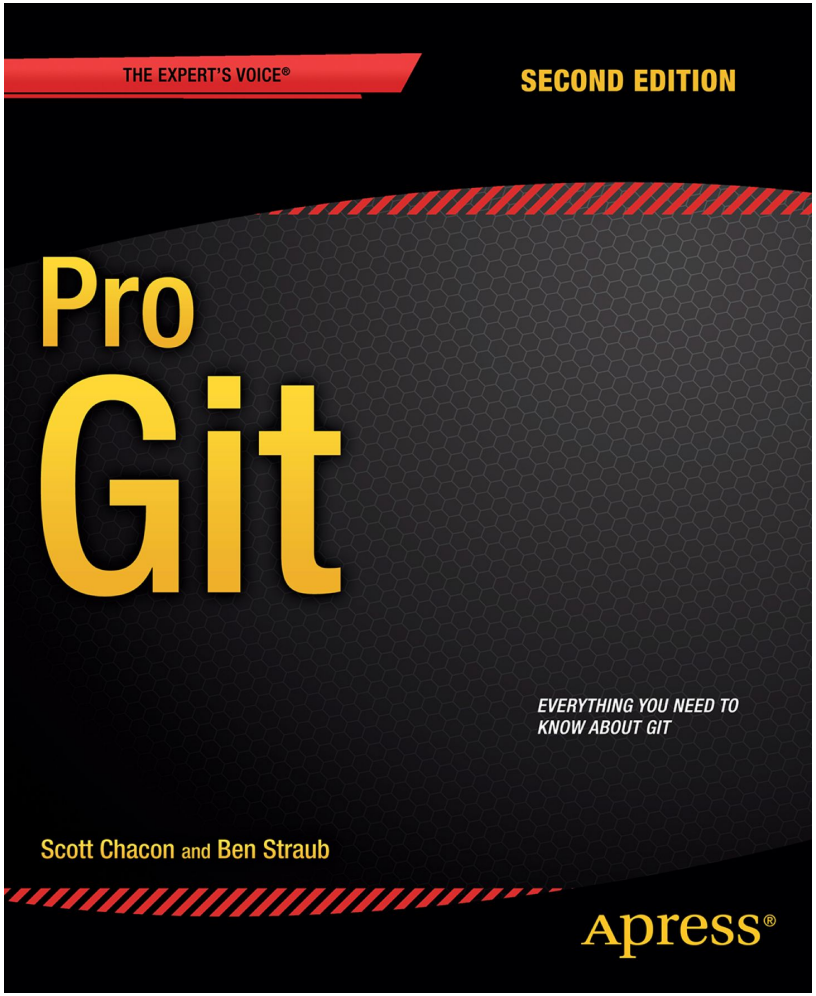


Source Control with Git

Assoc. Prof. Panche Ribarski, PhD | Assoc. Prof. Milos Jovanovik, PhD

Continuous Integration and Delivery 2023

Source Control with Git



https://git-scm.com/book/en/v2

Agenda

- Introduction
- Git: Overview
- Git: Getting Started
- Git: Branching
- Git: Collaboration Workflows on GitHub
- Conclusion
- Class Assignment / Homework

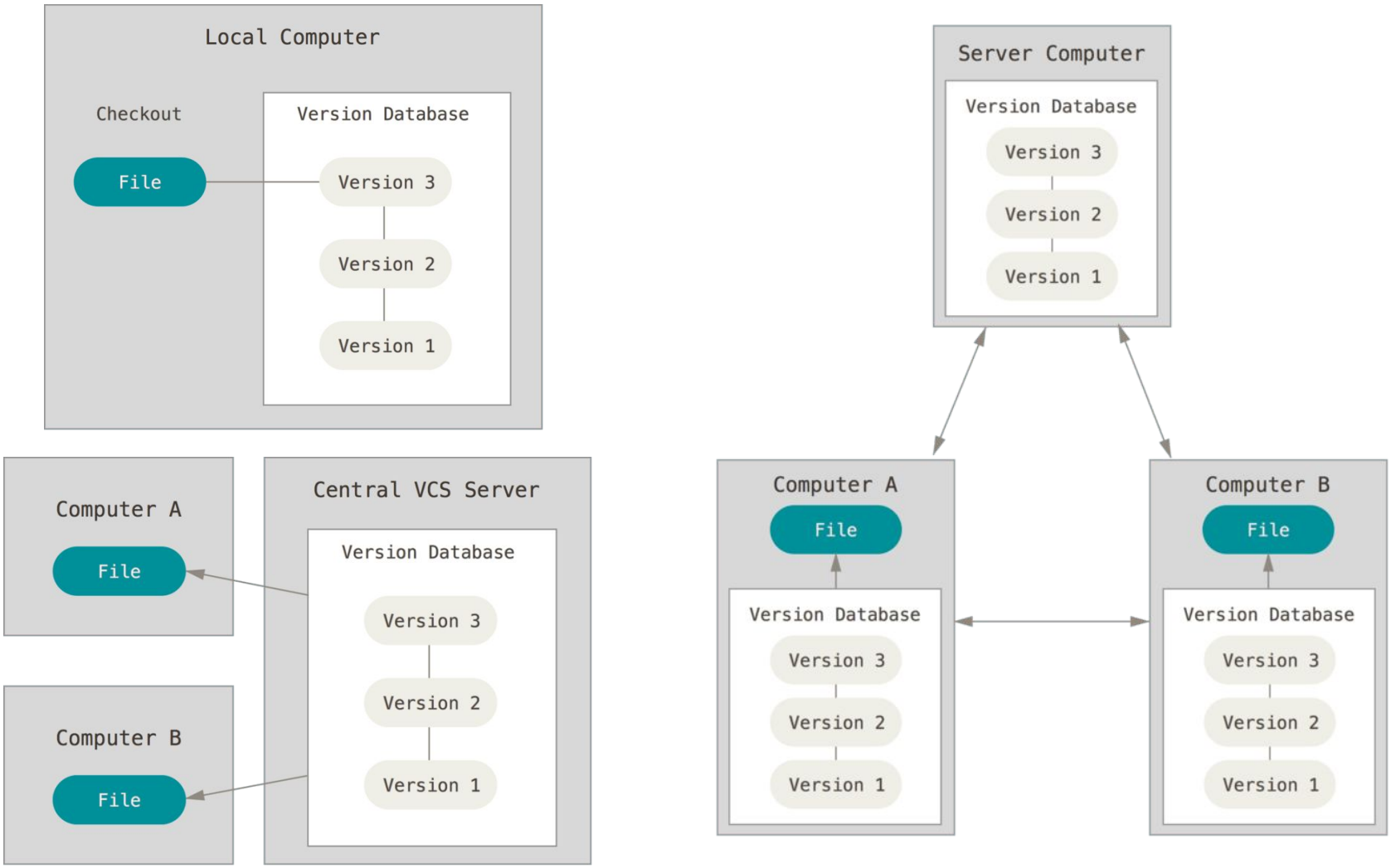
Introduction

- We've all done this
  - homework.pdf
  - homework-new.pdf
  - homework-newer.pdf
  - homework-newest.pdf
  - homework2.pdf
  - homework2-fix-new.pdf
  - homework-final.pdf
  - homework-final2.pdf

Introduction

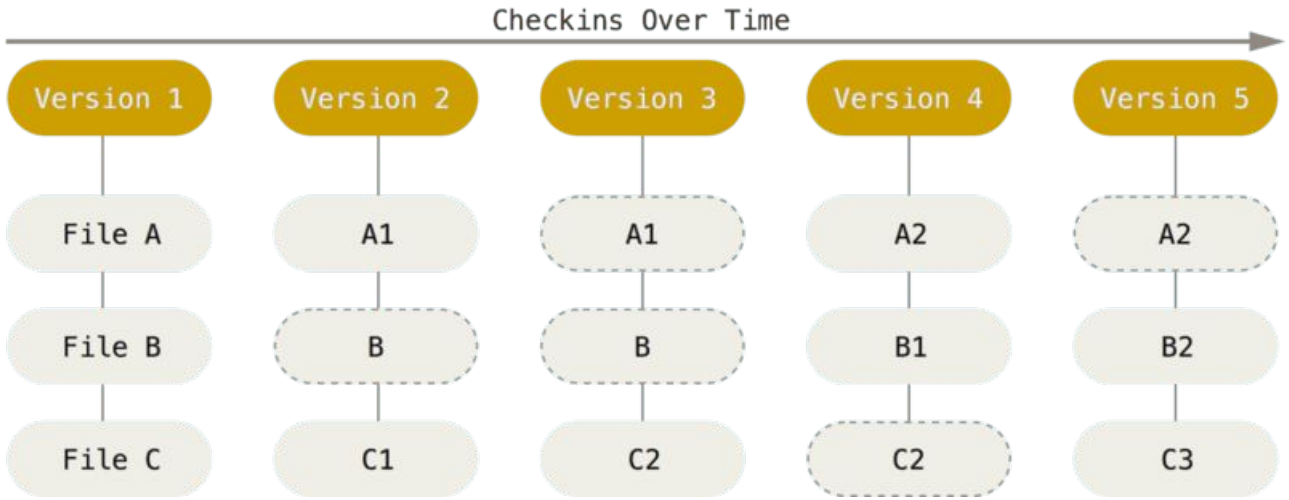
- Source Control (Version Control) is a vital component of the software development life-cycle.
  - Tracking changes to source code over time;
  - Collaborative work on the same codebase;
  - Detailed history of changes made to the codebase;
  - Possibility to revert to previous versions of the codebase;
  - Integrates with other software development tools: IDEs, build automation tools, CI/CD platforms, etc.;
  - Can also be used for other digital assets: documentation, configuration files, etc;
  - Popular SCM systems: Git, Subversion, Mercurial, Perforce, CVS, etc.

Types of VCSs: Local, Central, Distributed



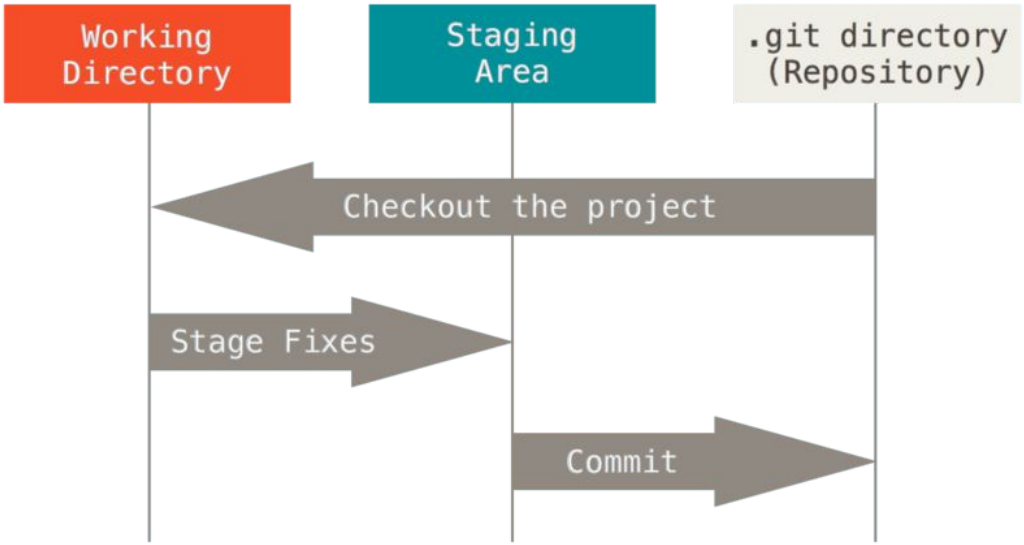
Git: Overview

- Git is a version control system (DVCS) used for tracking changes in source code during software development
  - Allows for a collaboration between multiple developers;
  - Create and work on different branches, merge changes, revert to previous versions, etc.;
  - Works like a stream of snapshots, not deltas;



Git: Overview

- Git has three main states that your files can reside in:
  - Modified: you have changed the file, but have not committed it to your database yet;
  - Staged: you have marked a modified file in its current version to go into your next commit snapshot;
  - Committed: the data is safely stored in your local database;



Git: Getting Started

- You typically obtain a Git repository in one of two ways:
  - you can take a local directory that is currently not under version control, and turn it into a Git repository; or
  - you can clone an existing Git repository from elsewhere;

- Initializing a repo in an existing directory: `git init`

```
debian@playground:~/kiii$ mkdir git-exercise-01
debian@playground:~/kiii$ cd git-exercise-01/
debian@playground:~/kiii/git-exercise-01$ git init
Initialized empty Git repository in /home/debian/kiii/git-exercise-01/.git/
debian@playground:~/kiii/git-exercise-01$
```


Git: Getting Started

- Staging files for the repo: `git add`
- Commiting files to the repo: `git commit`

```
debian@playground:~/kiii/git-exercise-01$ nano README
debian@playground:~/kiii/git-exercise-01$ git add README
debian@playground:~/kiii/git-exercise-01$ git commit -m 'Initial project version'
[master (root-commit) 56e9af6] Initial project version
 Committer: Debian <debian@playground.novalocal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 README
debian@playground:~/kiii/git-exercise-01$
```

Git: Getting Started

- Cloning an existing repository: `git clone`

```
debian@playground:~/kiii/git-exercise-02$ git clone https://github.com/
hobbit-project/ontology
Cloning into 'ontology'...
remote: Enumerating objects: 136, done.
remote: Total 136 (delta 0), reused 0 (delta 0), pack-reused 136
Receiving objects: 100% (136/136), 33.33 KiB | 975.00 KiB/s, done.
Resolving deltas: 100% (68/68), done.
debian@playground:~/kiii/git-exercise-02$ ls -l
total 4
drwxr-xr-x 4 debian debian 4096 Feb 21 13:00 ontology
debian@playground:~/kiii/git-exercise-02$ cd ontology/
debian@playground:~/kiii/git-exercise-02/ontology$ ls -l
total 36
-rw-r--r-- 1 debian debian   185 Feb 21 13:00 Makefile
-rw-r--r-- 1 debian debian   988 Feb 21 13:00 README.md
-rw-r--r-- 1 debian debian  1680 Feb 21 13:00 errors.ttl
drwxr-xr-x 2 debian debian  4096 Feb 21 13:00 example
-rw-r--r-- 1 debian debian 16672 Feb 21 13:00 ontology.ttl
debian@playground:~/kiii/git-exercise-02/ontology$
```


Git: Getting Started

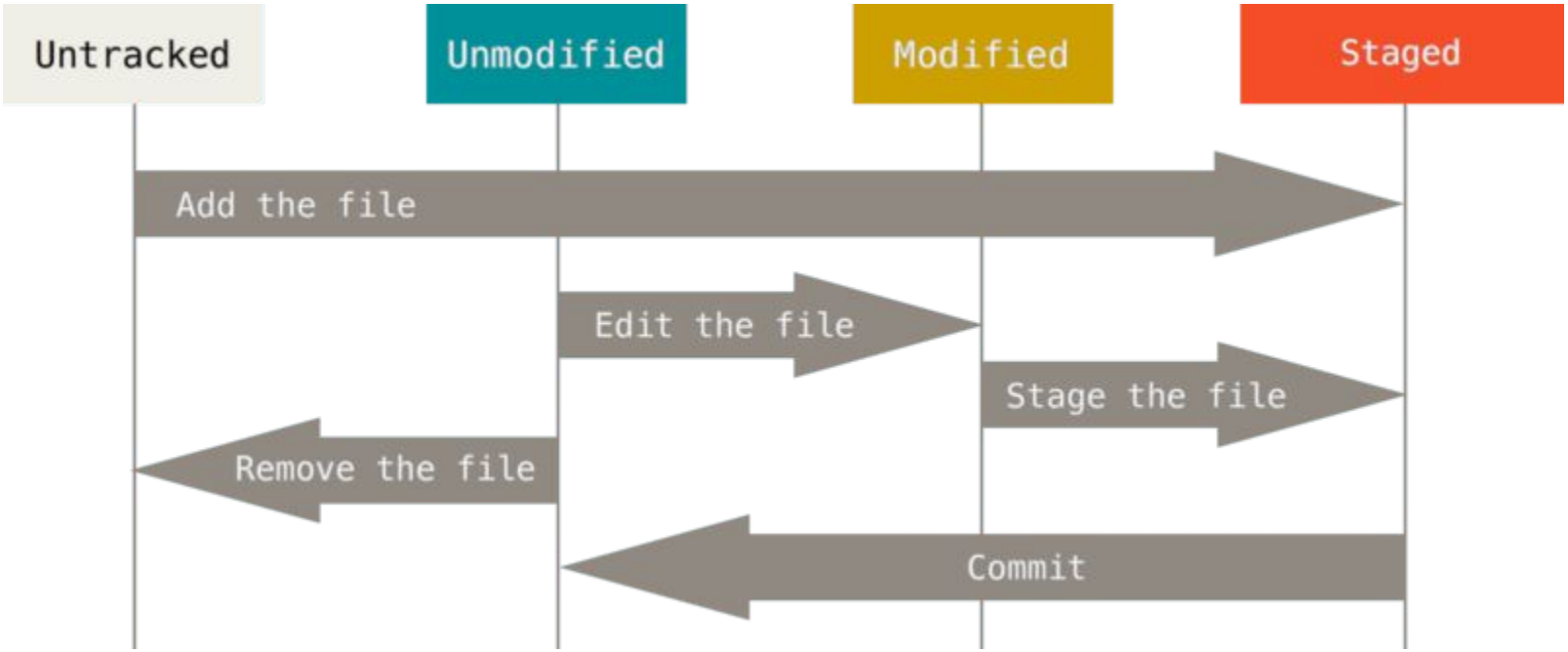
- Check the repo status: `git status`

```
debian@playground:~/kiii/git-exercise-03$ git clone https://github.com/
mjovanovik/kiii
Cloning into 'kiii'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
debian@playground:~/kiii/git-exercise-03$ cd kiii/
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
debian@playground:~/kiii/git-exercise-03/kiii$
```


Git: Getting Started

- The lifecycle of the status of your files



Git: Getting Started

- Tracking new files: `git add`

```
debian@playground:~/kiii/git-exercise-03/kiii$ echo "This is a sample
project" > README
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

        README

nothing added to commit but untracked files present (use "git add" to
track)
debian@playground:~/kiii/git-exercise-03/kiii$ git add README
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   README

debian@playground:~/kiii/git-exercise-03/kiii$
```


Git: Getting Started

- Staging modified existing files: `git add` (again)

```
debian@playground:~/kiii/git-exercise-03/kiii$ nano text.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   README

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:   text.txt

debian@playground:~/kiii/git-exercise-03/kiii$
```


Git: Getting Started

- Staging modified existing files: `git add` (again)

```
debian@playground:~/kiii/git-exercise-03/kiii$ git add text.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   README
        modified:   text.txt

debian@playground:~/kiii/git-exercise-03/kiii$
```


Git: Getting Started

- Short status report: `git status -s`

```
debian@playground:~/kiii/git-exercise-03/kiii$ git status -s
A  README
M  text.txt
debian@playground:~/kiii/git-exercise-03/kiii$
```


Git: Getting Started

- Viewing the changes: `git diff (--staged)`

```
debian@playground:~/kiii/git-exercise-03/kiii$ git diff
debian@playground:~/kiii/git-exercise-03/kiii$ git diff --staged
diff --git a/README b/README
new file mode 100644
index 0000000..2e271ed
--- /dev/null
+++ b/README
@@ -0,0 +1 @@
+This is a sample project
diff --git a/text.txt b/text.txt
index 6f3a977..8d42cd0 100644
--- a/text.txt
+++ b/text.txt
@@ -1 +1 @@
-This is a sample text file.
+This is a sample text file. This is a modification.
debian@playground:~/kiii/git-exercise-03/kiii$
```

Git: Getting Started

- Committing the changes: `git commit (-m)`

```
debian@playground:~/kiii/git-exercise-03/kiii$ git commit -m "First commit"
[main 72b99bb] First commit
 Committer: Debian <debian@playground.novalocal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 README
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Git: Getting Started

- Pushing the changes to GitHub: `git push`

```
debian@playground:~/kiii/git-exercise-03/kiii$ git push
Username for 'https://github.com': mjovanovik
Password for 'https://mjovanovik@github.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 389 bytes | 389.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/mjovanovik/kiii
   ea846cc..72b99bb  main -> main
debian@playground:~/kiii/git-exercise-03/kiii$
```

Git: Getting Started

- Moving a file: `git mv`

```
debian@playground:~/kiii/git-exercise-03/kiii$ git mv README README.md
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    README -> README.md

debian@playground:~/kiii/git-exercise-03/kiii$
```


Git: Getting Started

- Removing a file: `git rm`

```
debian@playground:~/kiii/git-exercise-03/kiii$ rm text.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    README -> README.md

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

        deleted:    text.txt
```

Git: Getting Started

- Removing a file: `git rm`

```
debian@playground:~/kiii/git-exercise-03/kiii$ git rm text.txt
rm 'text.txt'
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    README -> README.md
        deleted:    text.txt

debian@playground:~/kiii/git-exercise-03/kiii$
```


Git: Getting Started

- Ignoring files: The `.gitignore` file
- Used for files which are not needed in the repository:
  - log files, build files, temporary files, etc.
- Uses simplified regular expressions, just like shells do:
  - # . * ? [0-9] [abc] ...
- A large collection of `.gitignore` templates is available from GitHub:
  - https://github.com/github/gitignore

Git: Getting Started

Here is another example `.gitignore` file:

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

Git: Getting Started

- Undoing things: It can be a bit complicated
  - There are some undos that cannot be undone;
  - One of the few areas in Git in which you can lose data;
- Correcting a commit: `git commit --amend`
  - Can be used to correct a recent commit;
  - The previous commit is overridden: you end up with a new commit (with a new message, possibly);
- Unstaging a staged file: `git reset HEAD <file>`
  - Be careful, `reset` can be a dangerous command;
- Unmodifying a modified file: `git checkout -- <file>`
  - Discarding changes made to a file since last commit;

Git: Getting Started

- Correcting a commit: `git commit --amend`

```
debian@playground:~/kiii/git-exercise-03/kiii$ nano a.txt
debian@playground:~/kiii/git-exercise-03/kiii$ nano b.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git add a.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   a.txt

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

        b.txt

debian@playground:~/kiii/git-exercise-03/kiii$ git commit -m "Added a new fi
le"
[main 11a7736] Added a new file
 1 file changed, 1 insertion(+)
 create mode 100644 a.txt
```


Git: Getting Started

- Correcting a commit: `git commit --amend`

```
debian@playground:~/kiii/git-exercise-03/kiii$ git add b.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git commit --amend -m "Added
new files"
[main dc72a9d] Added new files
 Date: Sun Feb 26 17:47:34 2023 +0000
 2 files changed, 2 insertions(+)
 create mode 100644 a.txt
 create mode 100644 b.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git push
Enumerating objects: 5, done.
```


Git: Getting Started

- Unstaging a staged file: `git reset HEAD <file>`

```
debian@playground:~/kiii/git-exercise-03/kiii$ nano c.txt
debian@playground:~/kiii/git-exercise-03/kiii$ nano d.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git add *
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   c.txt
        new file:   d.txt
```


Git: Getting Started

- Unstaging a staged file: `git reset HEAD <file>`

```
debian@playground:~/kiii/git-exercise-03/kiii$ git reset HEAD d.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   c.txt

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

        d.txt

debian@playground:~/kiii/git-exercise-03/kiii$
```

Git: Getting Started

- Unmodifying a modified file: `git checkout -- <file>`

```
debian@playground:~/kiii/git-exercise-03/kiii$ nano d.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

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:   d.txt

no changes added to commit (use "git add" and/or "git commit -a")
debian@playground:~/kiii/git-exercise-03/kiii$ git checkout -- d.txt
debian@playground:~/kiii/git-exercise-03/kiii$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
debian@playground:~/kiii/git-exercise-03/kiii$
```


Git: Getting Started

- There is a new command `git restore`, introduced in Git version 2.23.0.
  - It's an alternative to `git reset`;
- Unstaging a staged file: `git restore --staged <file>`
- Unmodifying a modified file: `git restore <file>`
  - Be careful, this is also a dangerous command!

Git: Getting Started

- A list of other useful Git commands:
  - `git log`: viewing the commit history
  - `git config --global user.name "Name Surname"`: set Git user fullname
  - `git config --global user.name`: get Git user fullname
  - `git config --global user.email "user@provider.com"`: set Git user email
  - `git config --global user.email`: get Git user email
  - …

Git: Branching

- Branching - it means you diverge from the main line of development, and continue to do work without messing with that main line.

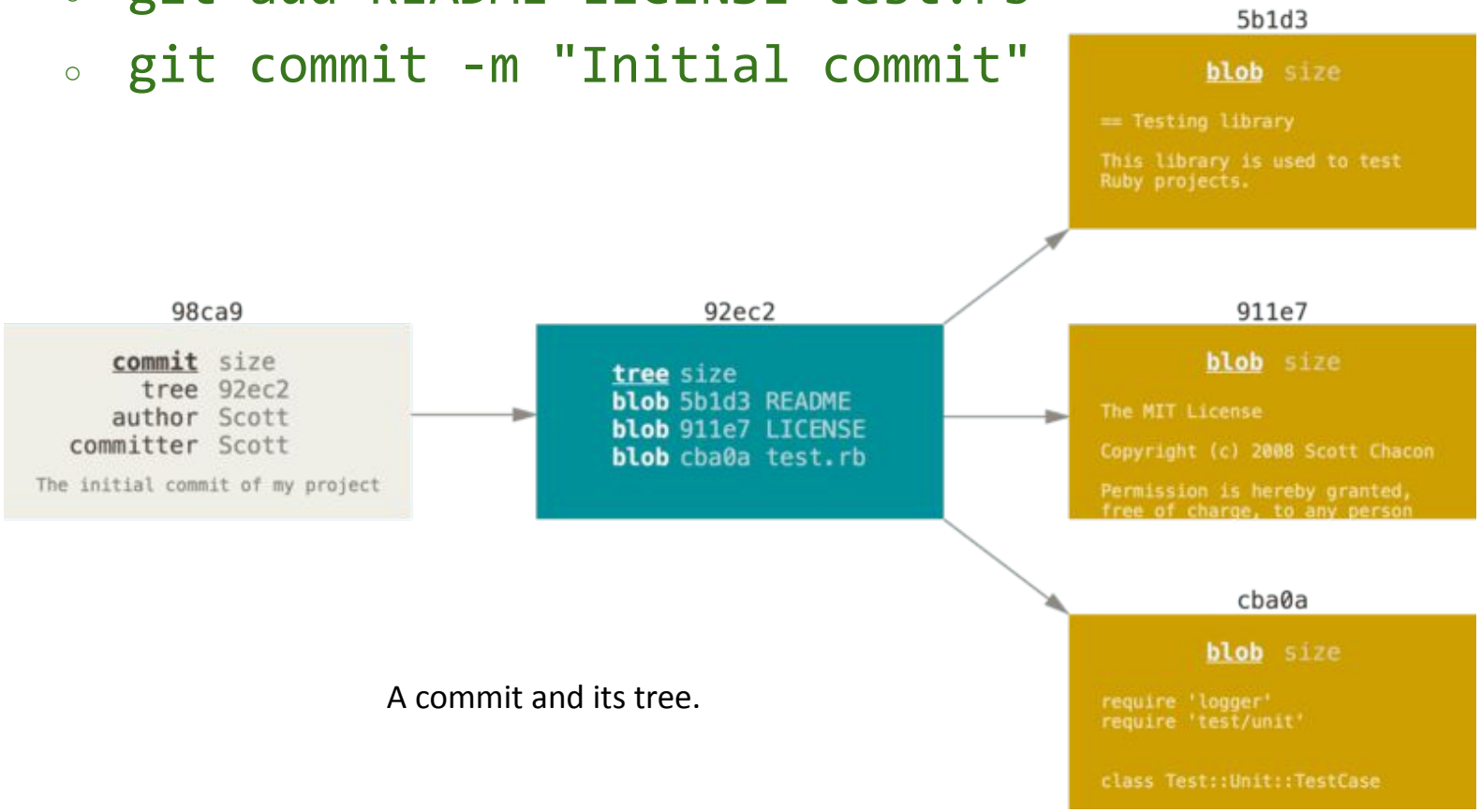
- Git branching is very lightweight.
  - Operations are nearly instantaneous;
  - Switching back-and-forth between branches is very fast;

Git: Branching

- How does it work? As we already stated, Git stores data as a series of snapshots.
- A Git commit creates a commit object which has:
  - A pointer to the snapshot of the content you staged;
  - The author's name and email address;
  - The message that you typed;
  - Pointer(s) to the commit(s) that directly came before this commit (its parent(s)):
    - 0 parents for the initial commit;
    - 1 parent for a normal commit;
    - 2+ parents for a commit that results from a merge of two or more branches;

Git: Branching

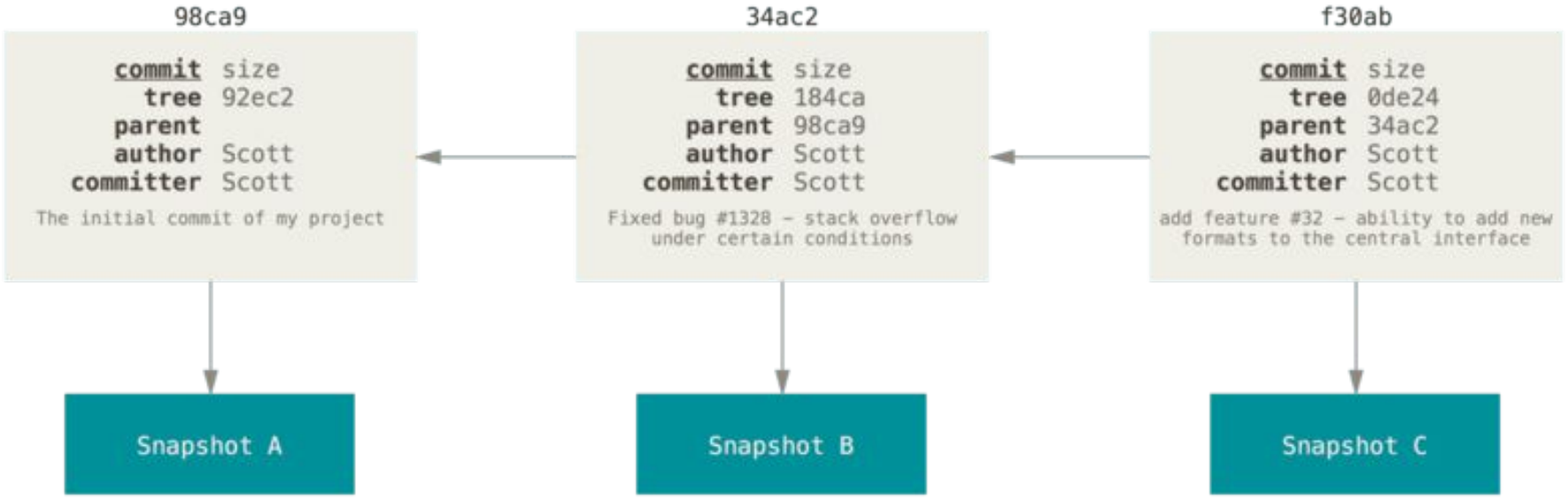
- Initial commit to a new repository.
  - `git add README LICENSE test.rb`
  - `git commit -m "Initial commit"`



A commit and its tree.

Git: Branching

- Each next commit is linked to their parent.

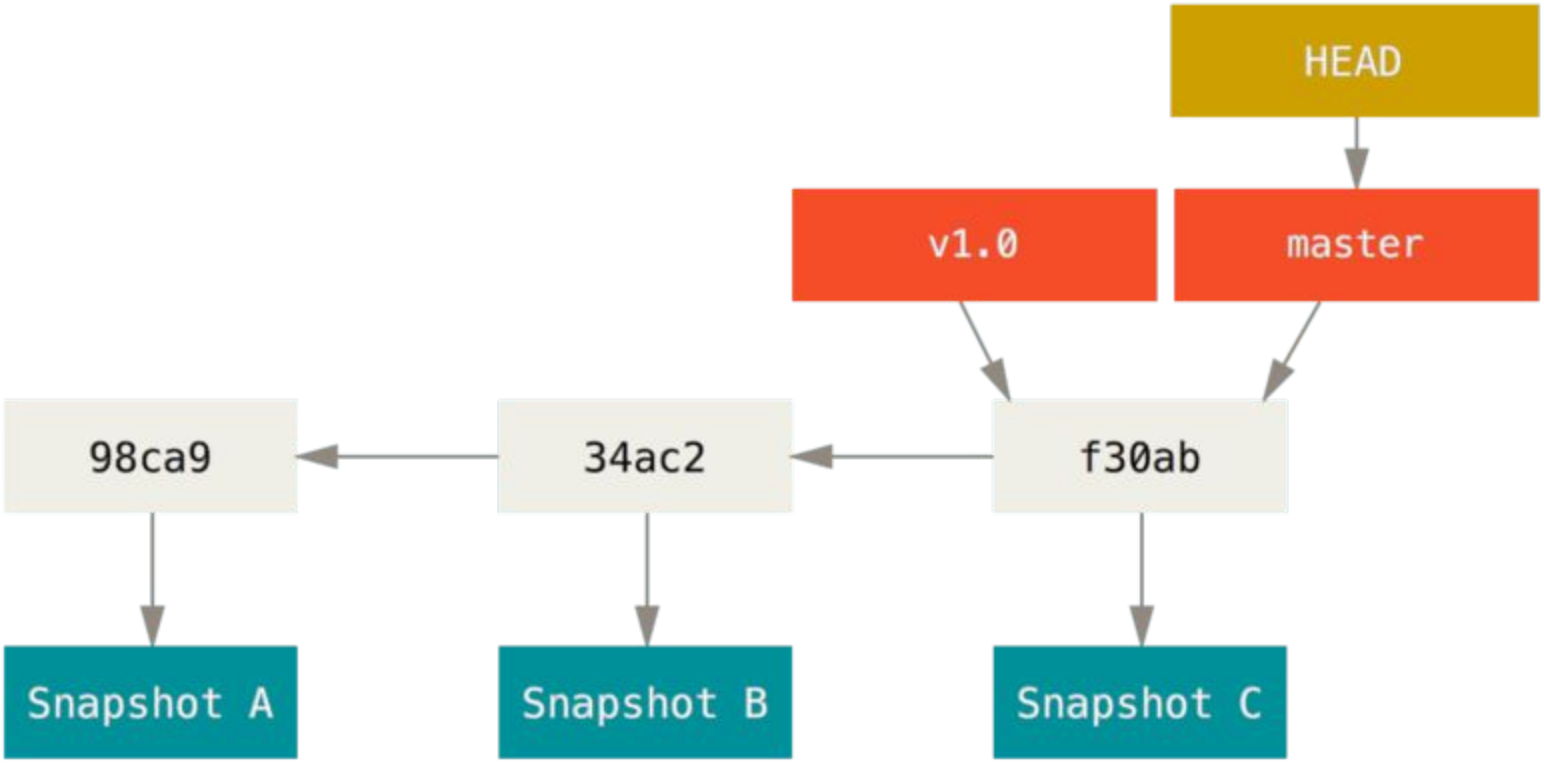


Commits and their parents.

Git: Branching

- Ok, so what is a branch?
  - A branch in Git is simply a lightweight movable pointer to one of these commits;
- The default branch name in Git is master*.
  - As you start making commits, you're given a master branch that points to the last commit you made;
  - Every time you commit, the master branch pointer moves forward automatically;
  - The master branch in Git is not a special branch – it is exactly like any other branch;
  - *Note: On GitHub, the default Git branch is called main;

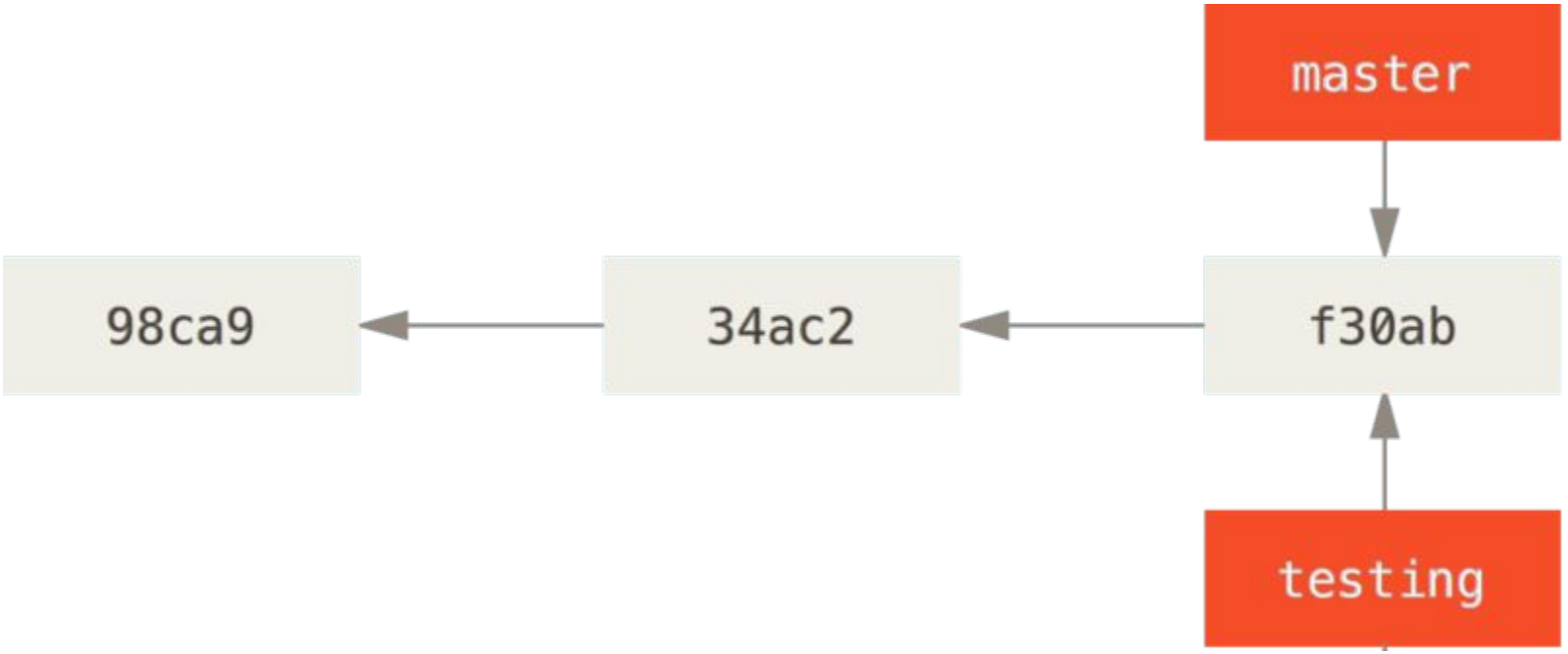
Git: Branching



A branch and its commit history.

Git: Branching

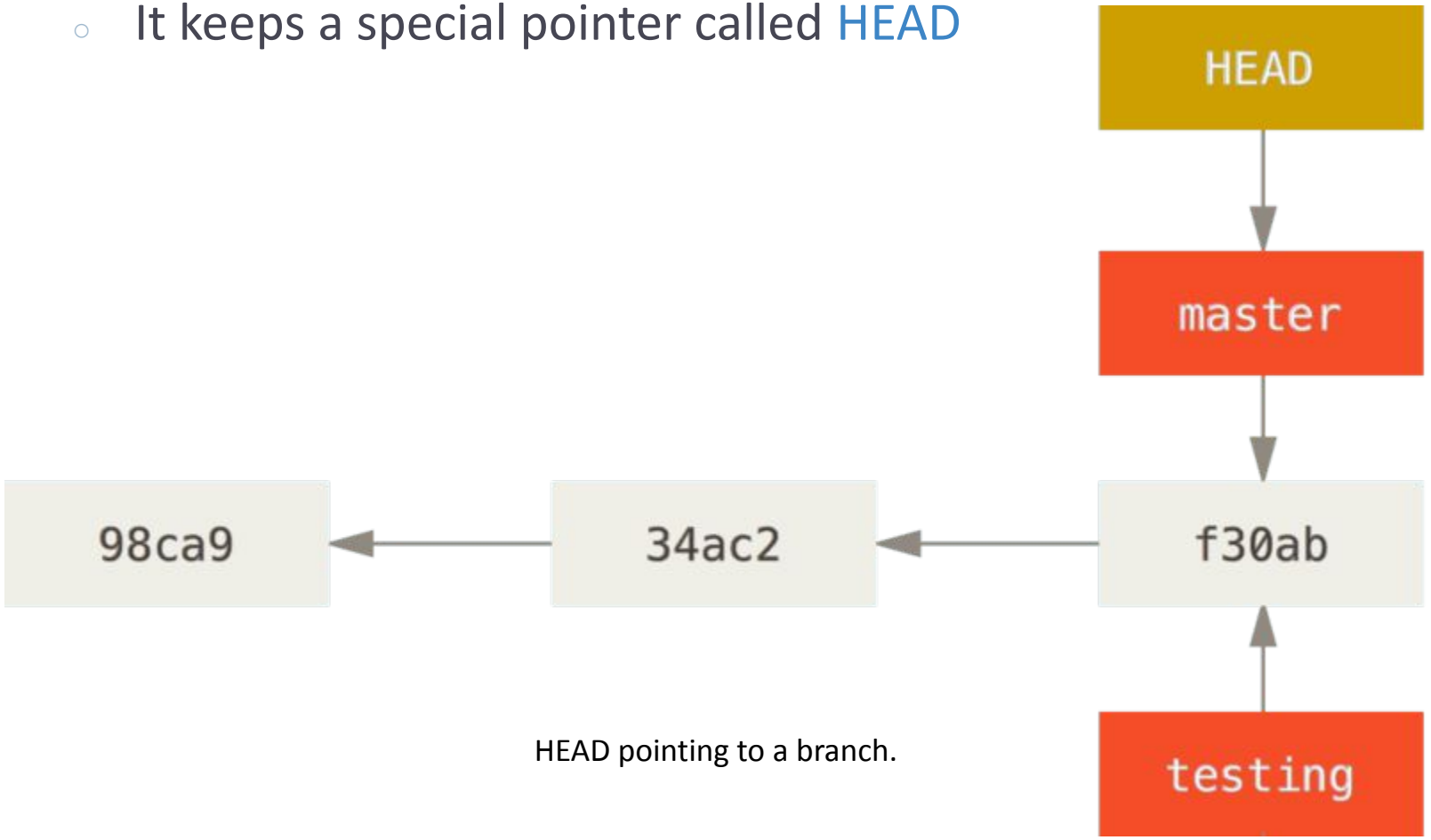
- Creating a new branch: `git branch`
  - Example: `git branch testing`



Two branches pointing into the same series of commits.

Git: Branching

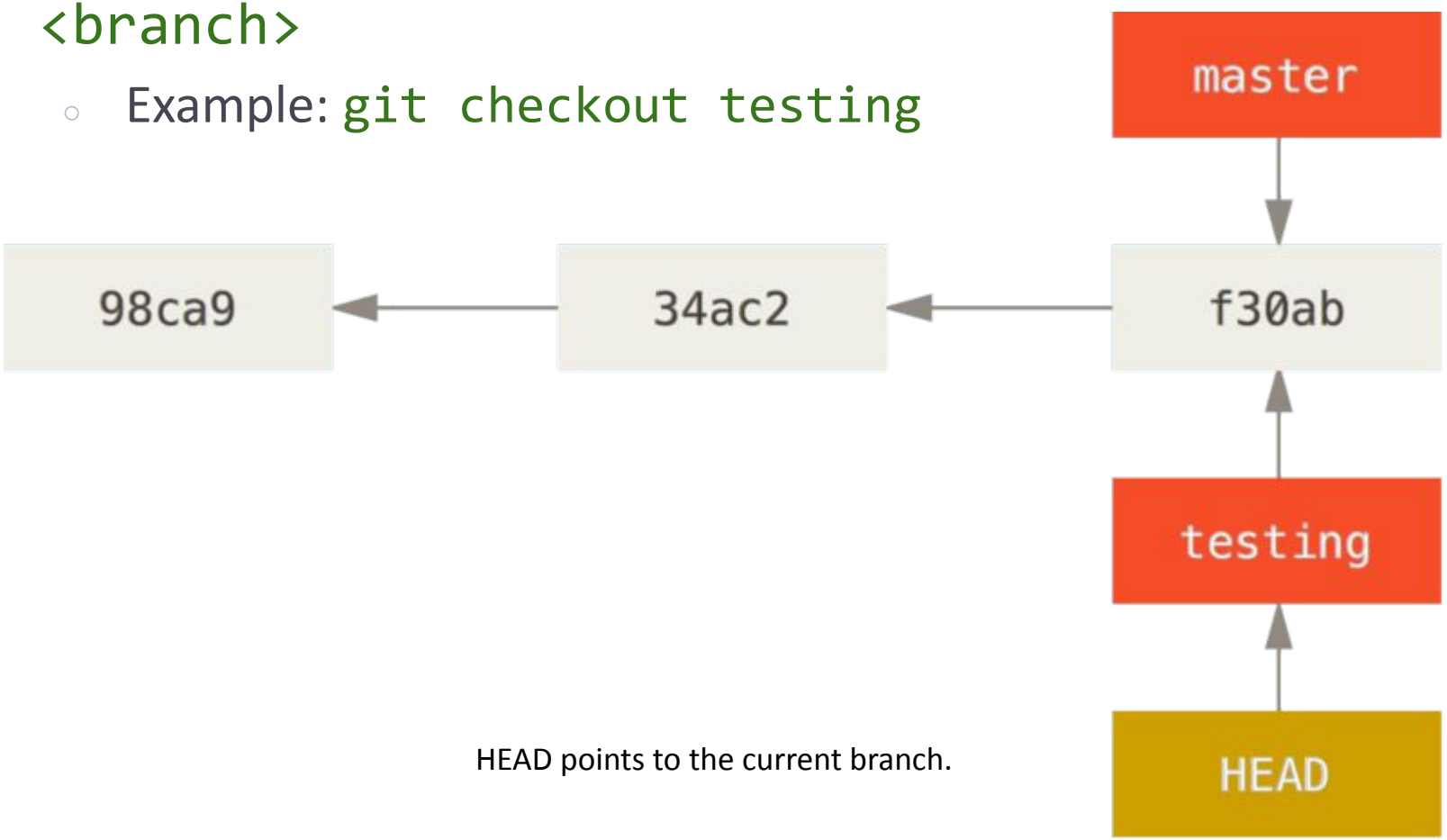
- How does Git know what branch you're currently on?
  - It keeps a special pointer called HEAD



HEAD pointing to a branch.

Git: Branching

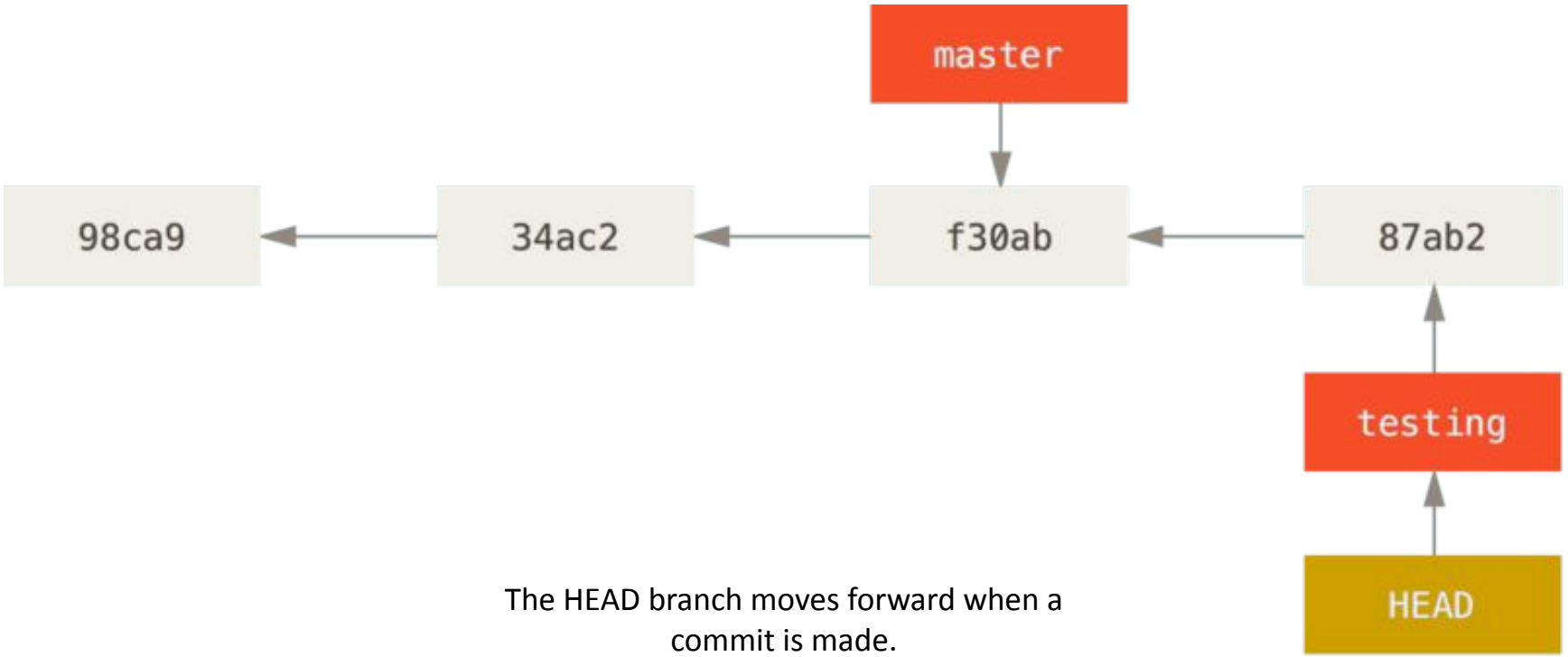
- Switching to another branch: `git checkout <branch>`
  - Example: `git checkout testing`



HEAD points to the current branch.

Git: Branching

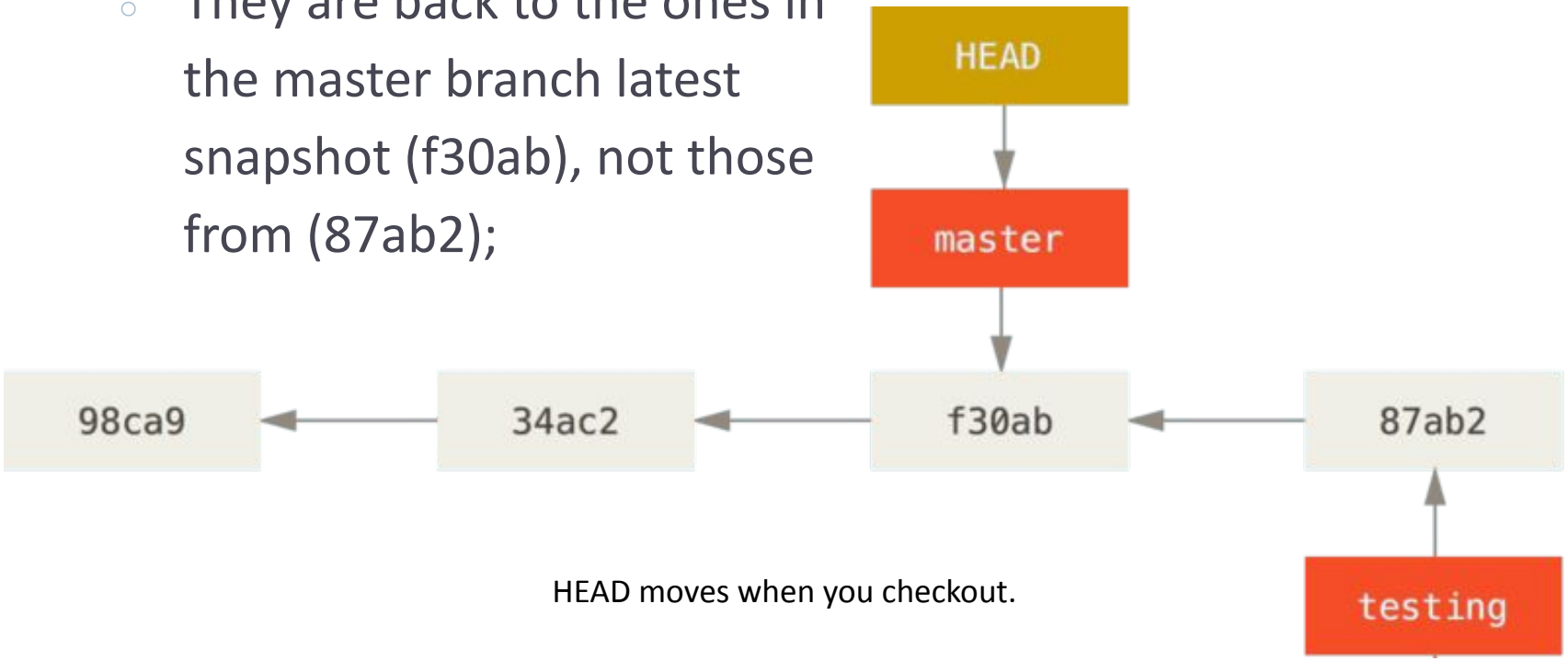
- Let's make some changes in the project repo:
  - `nano test.rb`
  - `git commit -a -m "made a change"`



The HEAD branch moves forward when a commit is made.

Git: Branching

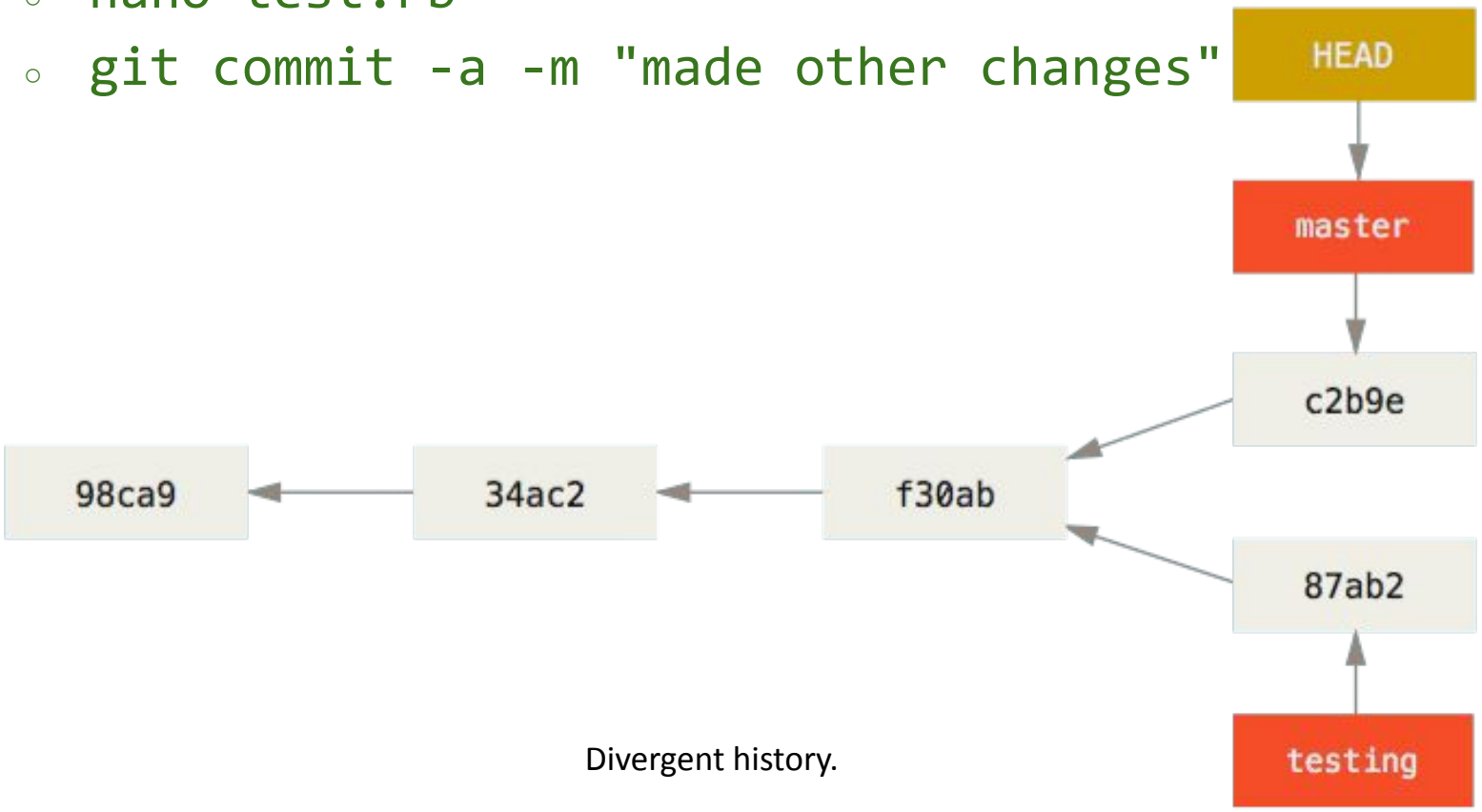
- Now, let's go back to the master branch:
  - `git checkout master`
- Be careful, this means your local files have changed!
  - They are back to the ones in the master branch latest snapshot (f30ab), not those from (87ab2);



HEAD moves when you checkout.

Git: Branching

- Let's make additional changes in the project repo:
  - `nano test.rb`
  - `git commit -a -m "made other changes"`



Divergent history.

Git: Branching

- These changes are visible in the logs:
  - `git log --oneline --decorate --graph --all`

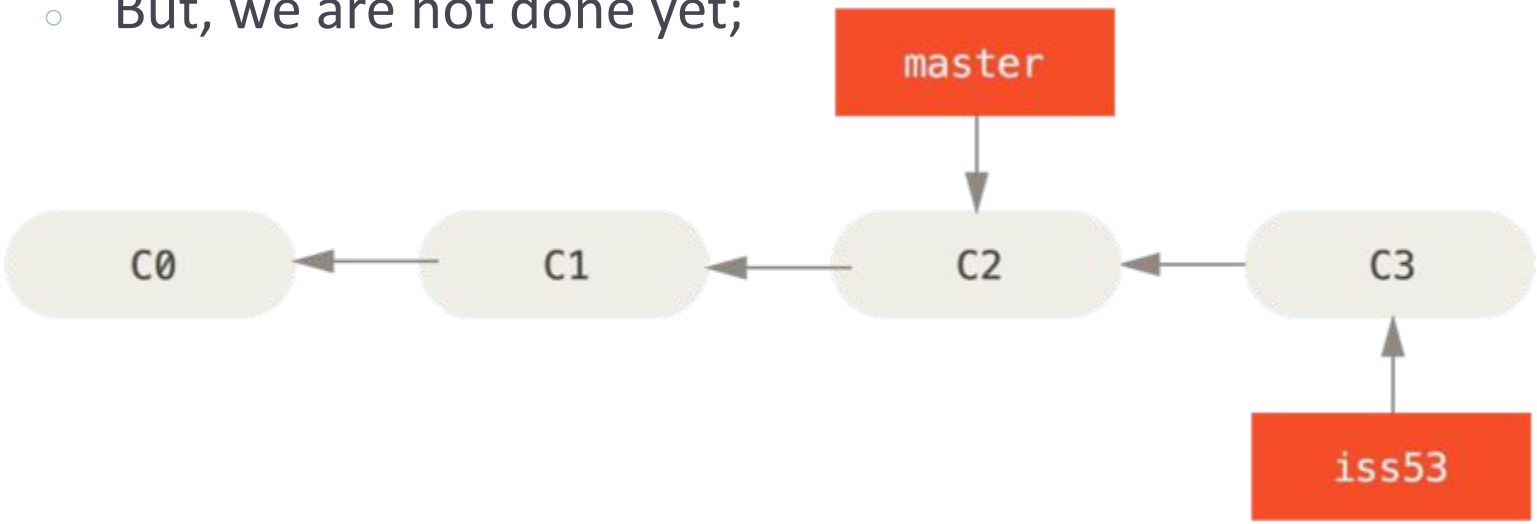
```
$ git log --oneline --decorate --graph --all
* c2b9e (HEAD, master) Made other changes
| * 87ab2 (testing) Made a change
|/
* f30ab Add feature #32 - ability to add new formats to the central interface
* 34ac2 Fix bug #1328 - stack overflow under certain conditions
* 98ca9 initial commit of my project
```

Git: Branching

- A branch in Git is actually a simple file:
  - It only contains the 40 character SHA-1 checksum of the commit it points to;
  - Therefore, branches are cheap to create and destroy;
  - Creating a new branch is as quick and simple as writing 41 bytes to a file (40 characters and a newline);
- A few notes:
  - Create a new branch and switch to it: `git checkout -b <newbranchname>`
  - New command since Git 2.23: `git switch`
    - `git switch <branchname>`
    - `git switch -c <newbranchname>`

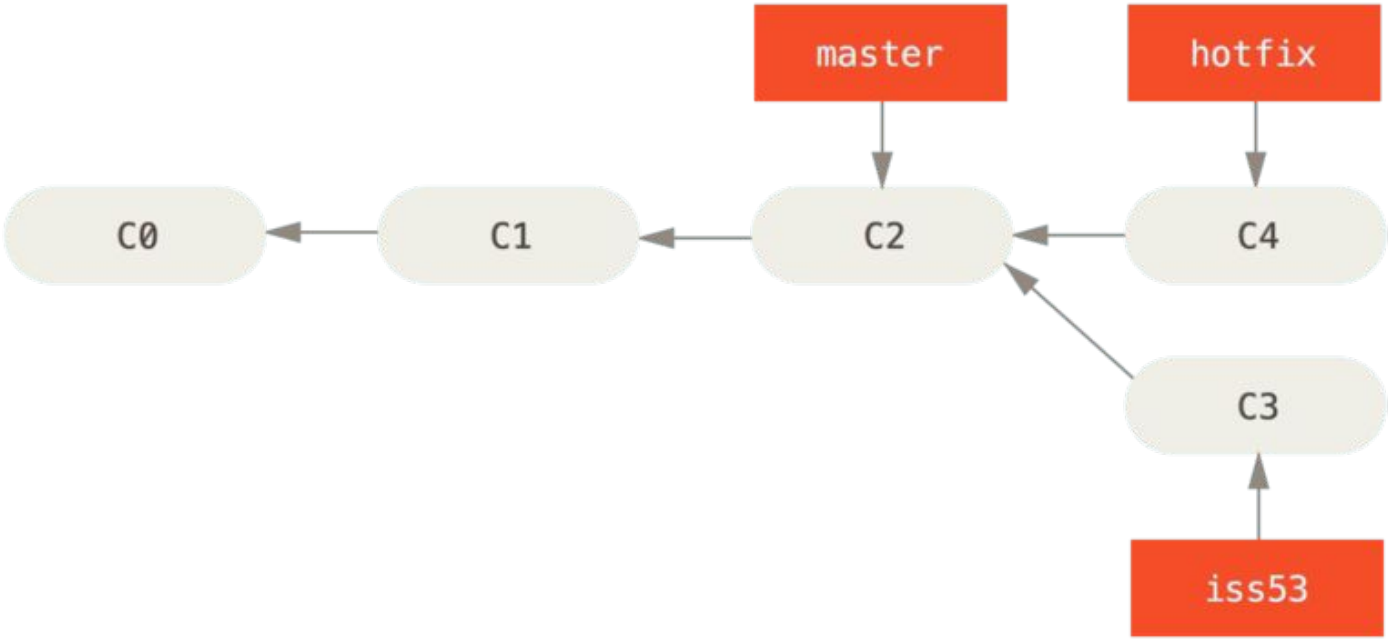
Git: Branching and Merging

- Let's look at a real-world scenario.
  - We have a software product, the master branch code works and is in production;
  - We've created a branch (iss53) to work on issue #53 in our project, and we've made some progress;
  - But, we are not done yet;



Git: Branching and Merging

- We get a message for an urgent fix to the master branch, because the client experienced a software bug.
  - We go back to the master branch (checkout);
  - We create a new branch in it (hotfix), for the quick bug fix;



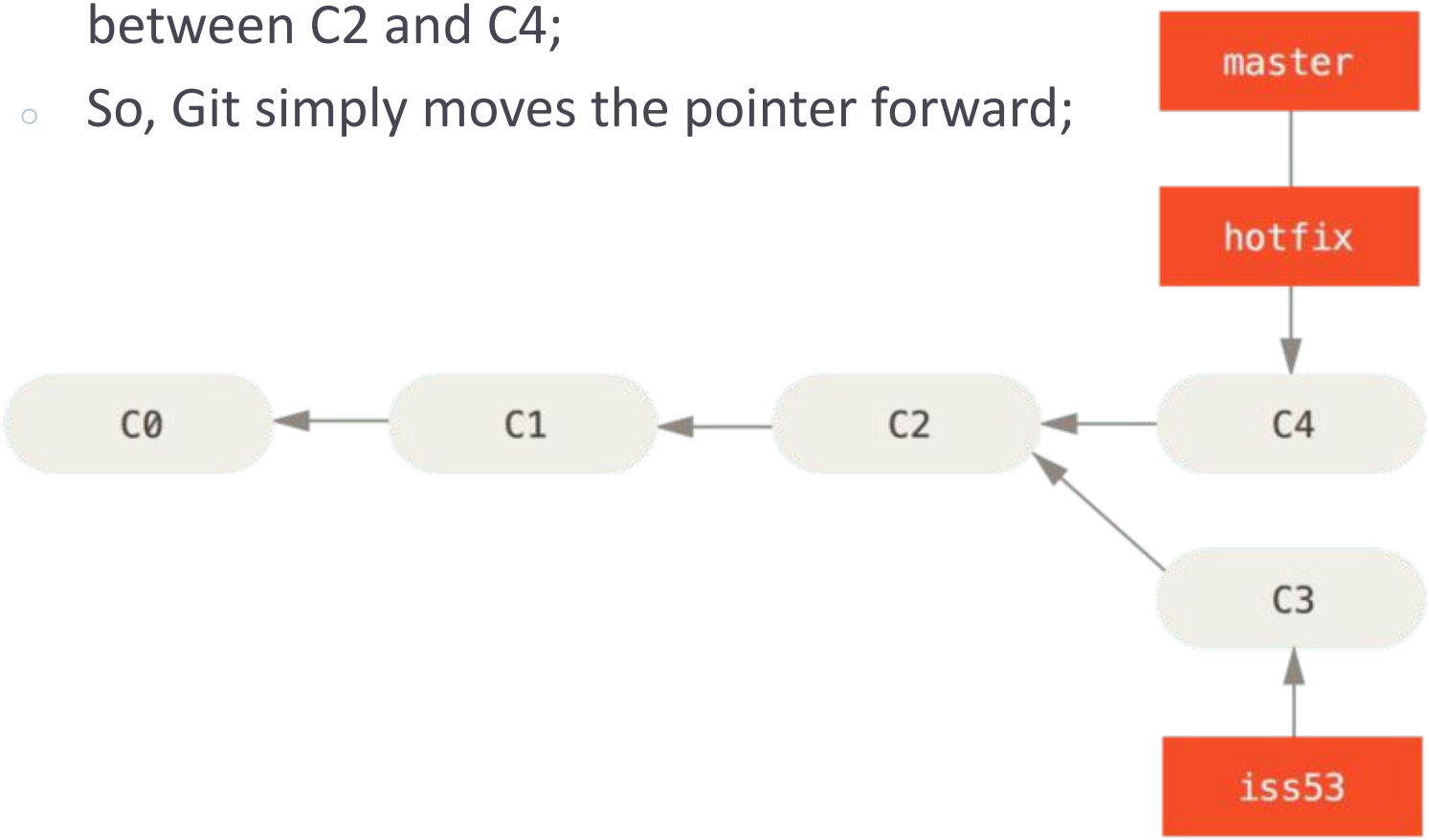
Git: Branching and Merging

- We finish with the fix in the hotfix branch, test that everything works as intended.
- We can now merge the hotfix branch back into the master branch.
  - `git branch master`
  - `git merge hotfix`

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```


Git: Branching and Merging

- Notice the phrase “fast-forward” in the merge.
  - This is a simple merge, because there is a direct line between C2 and C4;
  - So, Git simply moves the pointer forward;



Git: Branching and Merging

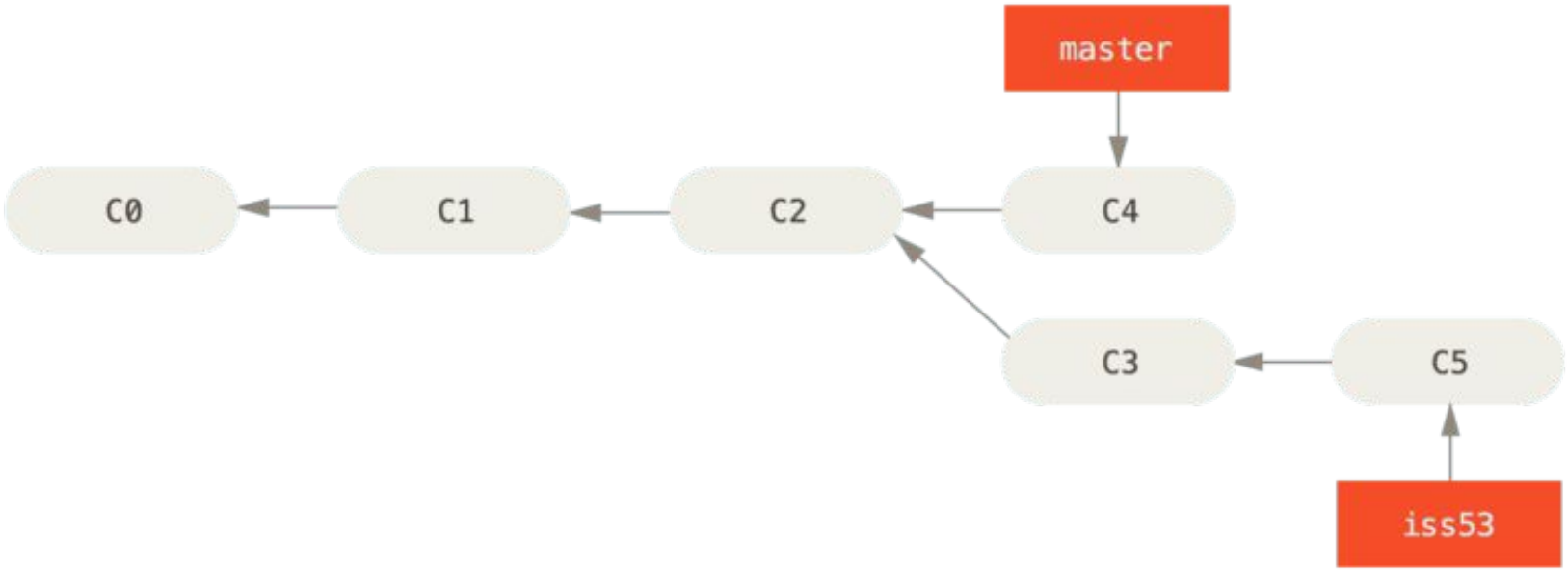
- Now you can go back to the iss53 branch, and continue the development.
- We can also delete the hotfix branch, now that master points to the same snapshot.

```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).
```

```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'Finish the new footer [issue 53]'
[iss53 ad82d7a] Finish the new footer [issue 53]
1 file changed, 1 insertion(+)
```


Git: Branching and Merging

- The new repository tree looks like this now:



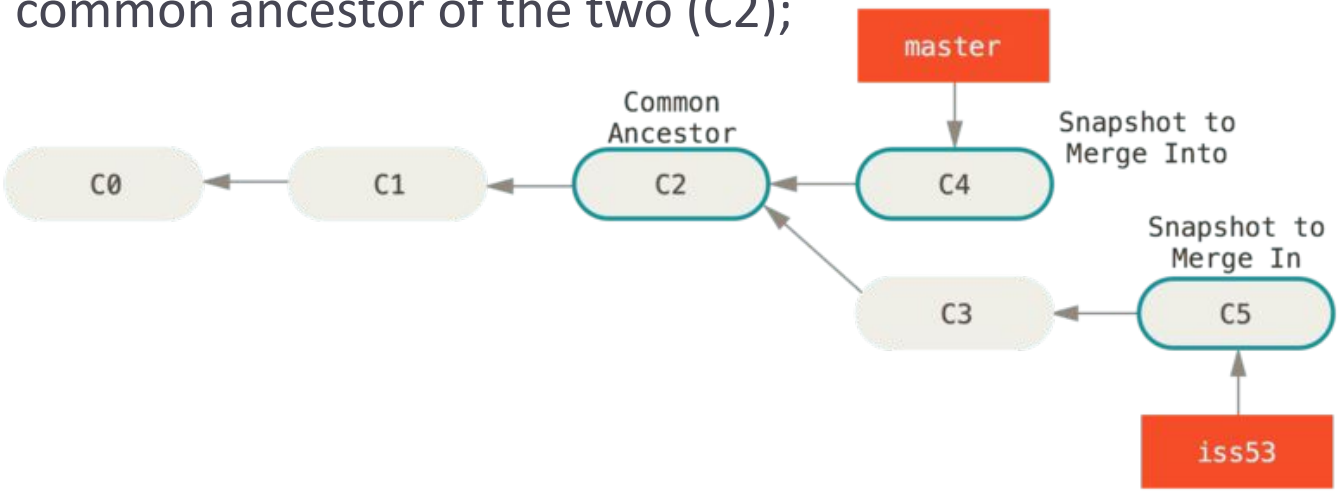
Git: Branching and Merging

- The work on issue #53 is now complete and ready to be merged into your master branch.
  - First, check out the branch you wish to merge into (master);
  - Then, run the `git merge` command:

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```

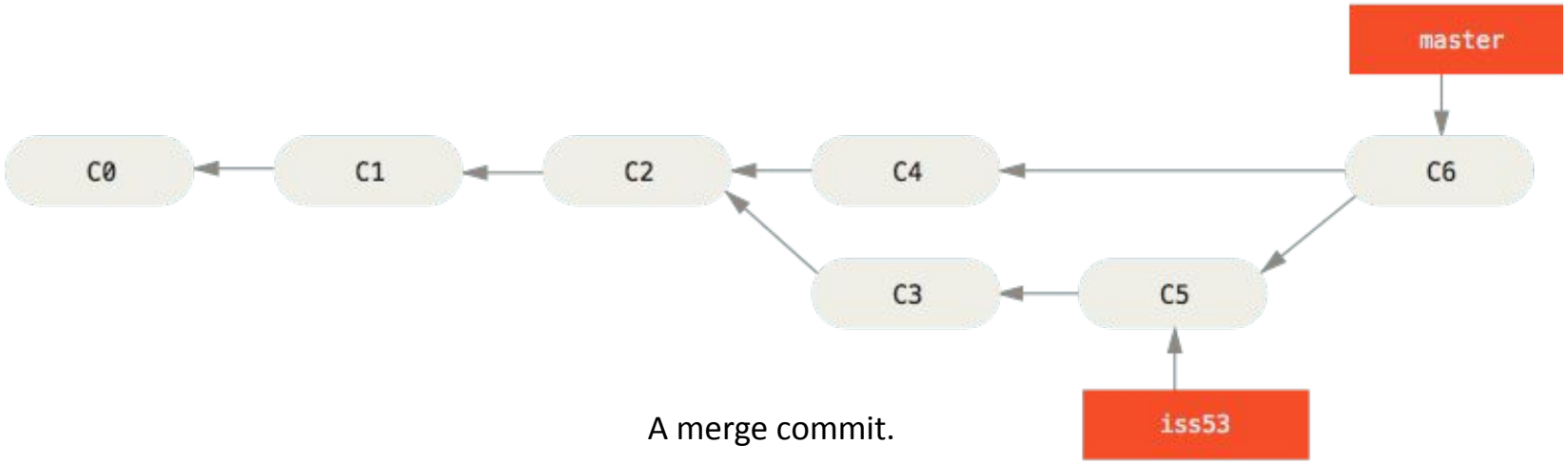
Git: Branching and Merging

- This looks a bit different than the hotfix merge from earlier, because here the development history has diverged.
  - Because the commit on the branch you're on (master) isn't a direct ancestor of the branch you're merging in (iss53), Git has to do some work;
  - In this case, Git does a simple three-way merge, using the two snapshots pointed to by the branch tips (C4 & C5) and the common ancestor of the two (C2);



Git: Branching and Merging

- Instead of just moving the branch pointer forward, Git creates a new snapshot that results from this three-way merge.
  - Then, it automatically creates a new commit that points to it;
  - This is referred to as a merge commit, and is special in that it has more than one parent;



A merge commit.

Git: Branching and Merging

- Merge Conflicts: Sometimes this process does not go smoothly.
  - If you changed the same part of the same file differently in the two branches you're merging, Git won't be able to merge them cleanly;

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Git: Branching and Merging

- The new merge commit failed and was paused for now.
  - You need to resolve the conflict, then run `git commit` manually;

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

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

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```


Git: Branching and Merging

- Git adds conflict-resolution markers to the files which have conflicts, so that you can see the problematic parts and correct them.

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

The correction.

```
<div id="footer">
please contact us at email.support@github.com
</div>
```

Git: Branching and Merging

```
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

    modified:   index.html
```

```
Merge branch 'iss53'

Conflicts:
    index.html
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   index.html
#
```


Collaboration Workflows on GitHub

- Forking Projects: If you want to contribute to an existing project on GitHub, to which you don't have push access, you can fork the project.
  - When you fork a project, GitHub will make a copy of the project that is entirely yours;
  - It lives in your namespace, and you can push to it;
  - When you're done with updates and changes, you can contribute your changes back to the original repository by creating what's called a Pull Request;
  - The Pull Request opens up a discussion thread with code review, and the owner and the contributor can communicate about the change until the owner is happy with it, at which point the owner can merge it in;

Collaboration Workflows on GitHub

- The GitHub Flow
  - Fork the project.
  - Create a topic branch from master.
  - Make some commits to improve the project.
  - Push this branch to your GitHub project.
  - Open a Pull Request on GitHub.
  - Discuss, and optionally continue committing.
  - The project owner merges or closes the Pull Request.
  - Sync the updated master back to your fork.

Conclusion

- Git is an essential tool for implementing DevOps practices in software development, enabling collaboration, automation, and continuous integration and deployment.
- With Git, developers can work on different branches of a codebase, allowing for parallel development and easy integration of changes.
- Git integrates seamlessly with other DevOps tools, such as continuous integration and deployment pipelines, enabling fast and reliable software delivery.
- By mastering Git and incorporating it into their DevOps workflows, developers can improve the quality, reliability, and efficiency of their software development processes.

Class Assignment / Homework

- Part 1: Working with your own Git repository
  - Create a GitHub account. Create an SSH key pair and add it to GitHub. Alternatively, create a personal access token.
  - Create a new GitHub repository and clone it locally.
  - Work locally to add new content, stage the files, make a commit, and push the changes to GitHub.
  - Create develop, test and production branches.
  - Make several changes and commit to the develop branch, then merge it into the other two branches. Push the changes to GitHub.
  - When you're done with the branch, merge it to main. Then, commit and push to GitHub.

Class Assignment / Homework

- Part 2: Contributing to another Git repository on GitHub
  - Fork a colleague's GitHub repository.
  - Create a new branch for a new feature.
  - Make changes, commit and push the changes to your GitHub repo.
  - When you're done, make a Pull Request with the new feature, via GitHub's website.
  - Ask your colleague to merge the Pull Request and close the issue.

Further Reading: Git Flows

- Git Flow
  - https://nvie.com/posts/a-successful-git-branching-model/
- GitHub Flow
  - https://guides.github.com/introduction/flow/
- GitLab Flow
  - https://docs.gitlab.com/ee/topics/gitlab_flow.html

Questions?