From zero to... Somewhat over zero: Git

Presented By: Ryan Schanzenbacher

RITLUG

ritlug.com

What is Git??

- Git is a free and open source VCS (Version Control System) developed by Linus T.
- Yeah, that Linus T. (Torvalds)

What is Git??

- Created by Torvalds in 2005 to replace the BitKeeper Source Control Management Utility
    - It was the best available tool at the time compared to others
    - The licensing model was **very** weird. It was proprietary, but use for certain OSS projects were allowed.
    - .....As long as you didn’t develop competing software
    - Which no one did! Not explicitly at least. The license to use BitKeeper was revoked when the owner of BitKeeper determined that Andrew Tridgell broke this license by reverse-engineering the protocol to make a client for BitKeeper to view metadata, such as diffs between commits.
    - This feature was only available on the commercial version of BitKeeper

What is Git??

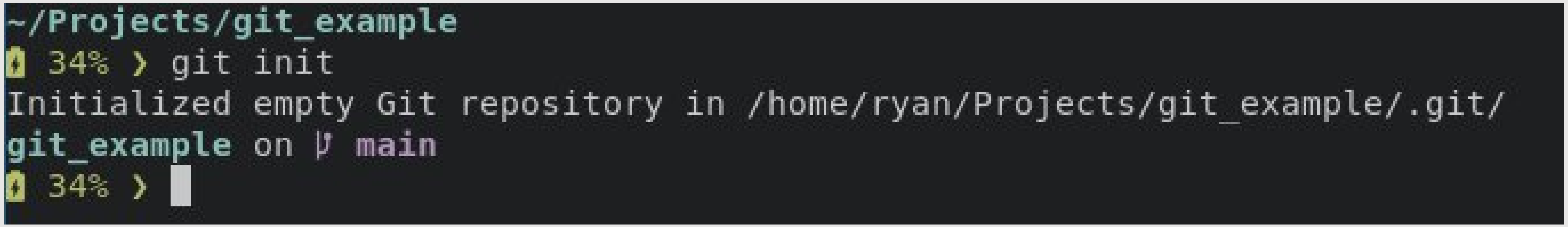
- Other solutions existed at the time BitKeeper access was removed (CVS, Subversion, etc.)
  - However one design goal of git was to take what CVS did as an example of what **NOT** to do. When in doubt, do the opposite!
- None of the options really pleased Linus in terms of performance and usability
  - He wanted merges to take no longer than 3 seconds, even in more complicated setups

What is Git??

- So, as a solution, he decided to make his own VCS.
    - .....As one does
- Development started on April 3, 2005
- Project was announced April 6, 2005
- Became self-hosting (meaning it used itself to track development) April 7, 2005
- Linux kernel switched over and first merge occurred April 18, 2005

Let’s create a repository!

- The beginning of any git repository is a `git init` command
  - This will transform the current working directory into the root of a git repository
  - Nothing is stored in the repo yet, but it can have properties (like branch names, remotes, etc.) added

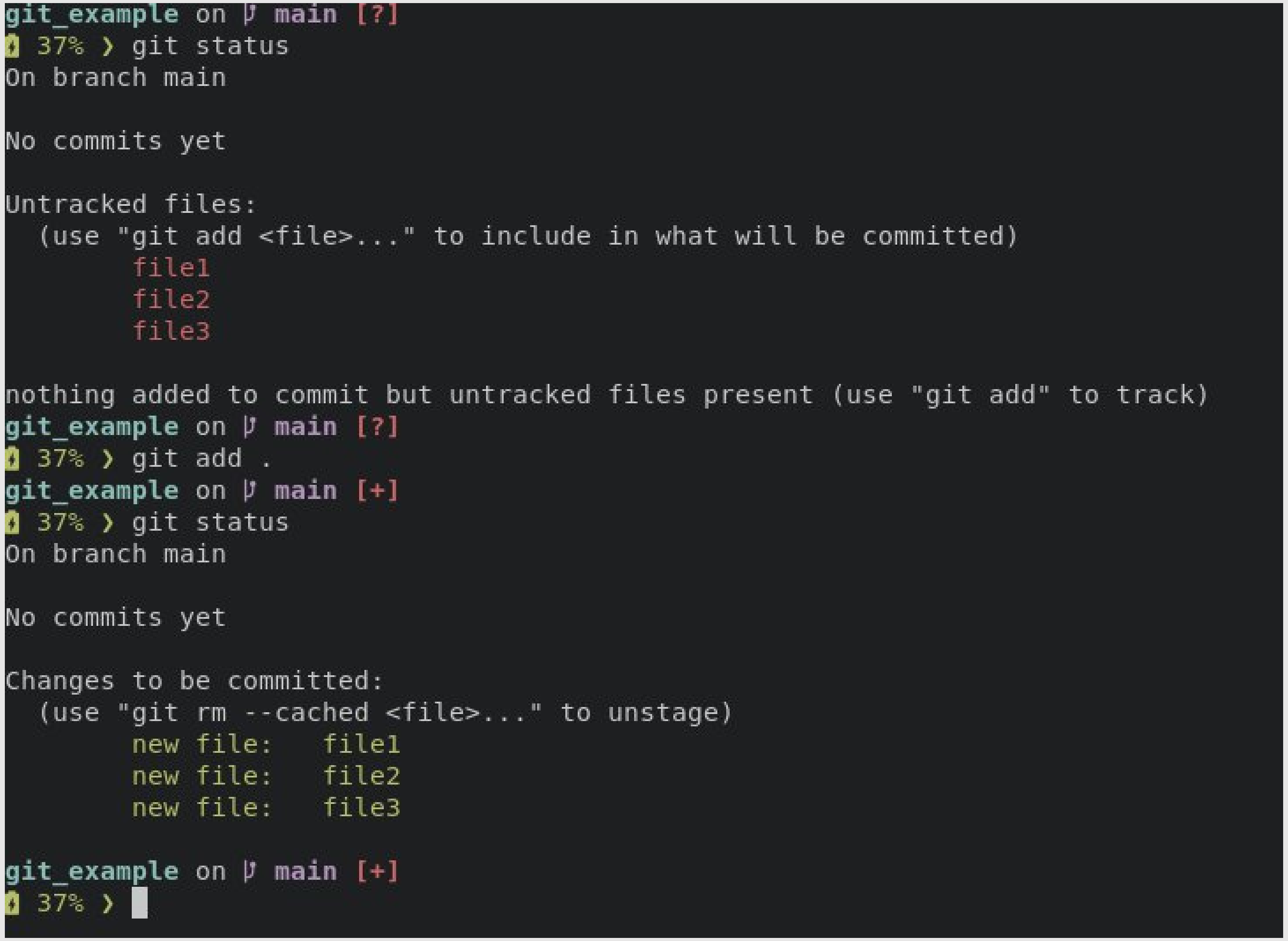


```
~/Projects/git_example
 34% ❯ git init
Initialized empty Git repository in /home/ryan/Projects/git_example/.git/
git_example on  main
 34% ❯
```

Let’s make some files

- Now that we have an empty (or not so empty) repository, let’s start tracking its state with git.
- The command `git add {filename}` will add files to git, making them ready to be commited in the next step
- You can add single files at a time to pick and choose what is included in a commit, or another common command is `git add .` for adding all changed files from PWD downwards

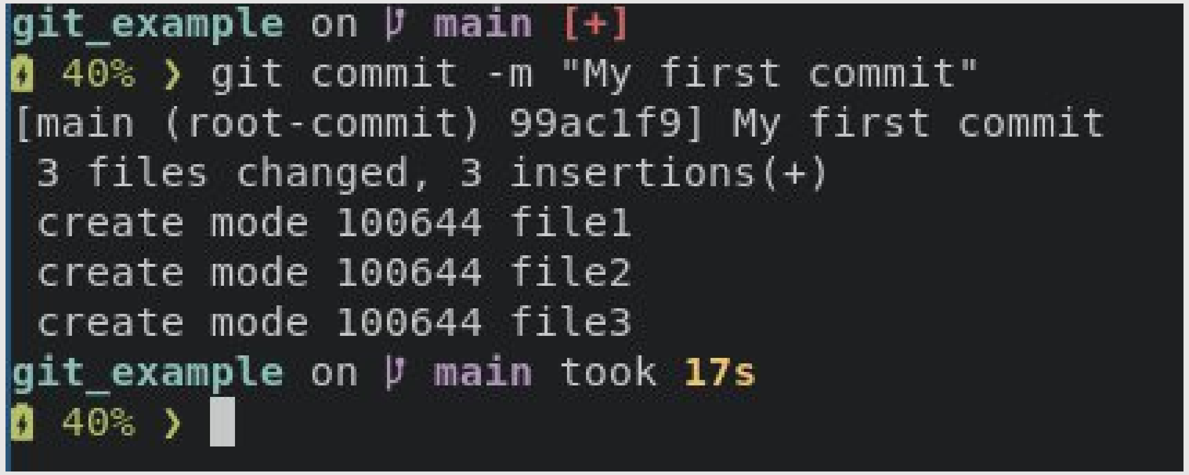
`git add .`



```
git_example on  main [?]
 37% ❯ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1
        file2
        file3

nothing added to commit but untracked files present (use "git add" to track)
git_example on  main [?]
 37% ❯ git add .
git_example on  main [+]
 37% ❯ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1
        new file:   file2
        new file:   file3

git_example on  main [+]
 37% ❯
```

Making it permanent

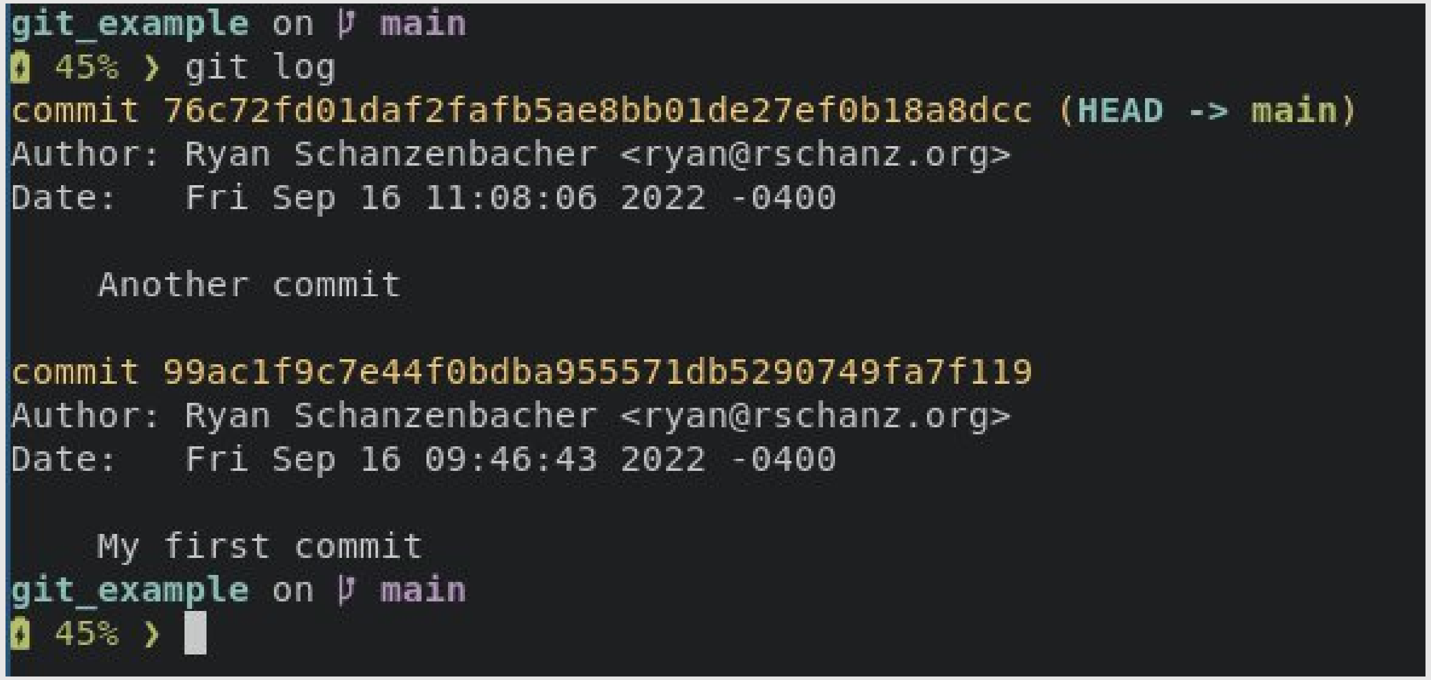
- The last step in the basic workflow is to commit your changes
- This will take a “snapshot” of your repo at that moment that you can reference back to later
- Can add messages, sign them with a GPG Key, and many other things



```
git_example on  main [+]
 40% ❯ git commit -m "My first commit"
[main (root-commit) 99ac1f9] My first commit
 3 files changed, 3 insertions(+)
 create mode 100644 file1
 create mode 100644 file2
 create mode 100644 file3
git_example on  main took 17s
 40% ❯
```

How do we find these again?

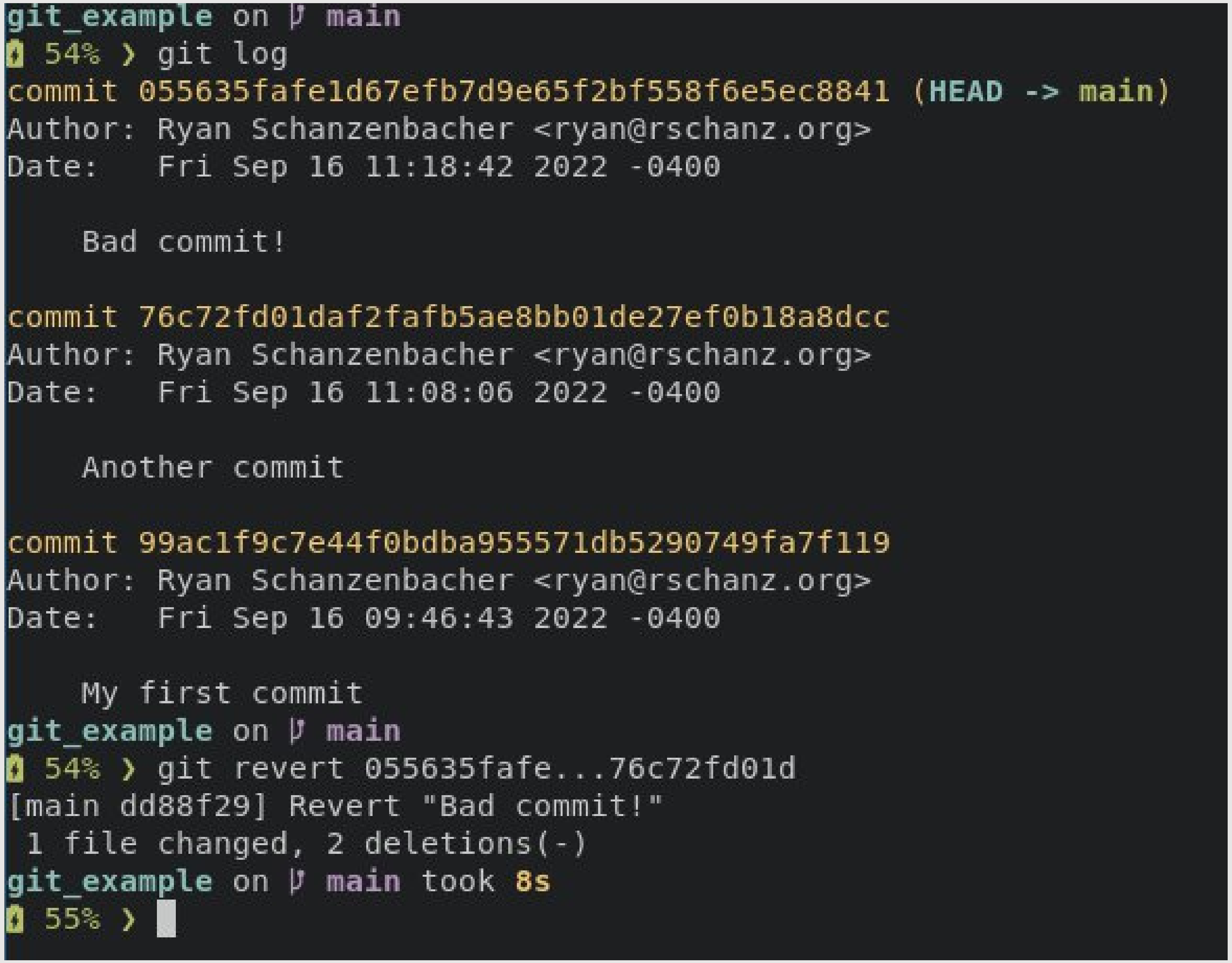
- Now that we have a commit made, we can view it (and the history of all our other commits) using the `git log` command



```
git_example on ᚠ main
 45% ❯ git log
commit 76c72fd01daf2fafb5ae8bb01de27ef0b18a8dcc (HEAD -> main)
Author: Ryan Schanzenbacher <ryan@rschanz.org>
Date:   Fri Sep 16 11:08:06 2022 -0400

    Another commit

commit 99ac1f9c7e44f0bdba955571db5290749fa7f119
Author: Ryan Schanzenbacher <ryan@rschanz.org>
Date:   Fri Sep 16 09:46:43 2022 -0400

    My first commit
git_example on ᚠ main
 45% ❯
```

Cool, now my code is messed up

- How do I actually use git to restore a previous version of my code?
- Use `git revert` to create a new commit that reverses previous commits
- `git revert` is the best option to reverse your entire repository, especially if you’ve already pushed to a remote (like github) since you will avoid force-pushing and ruining history.
- Other option DO exist!



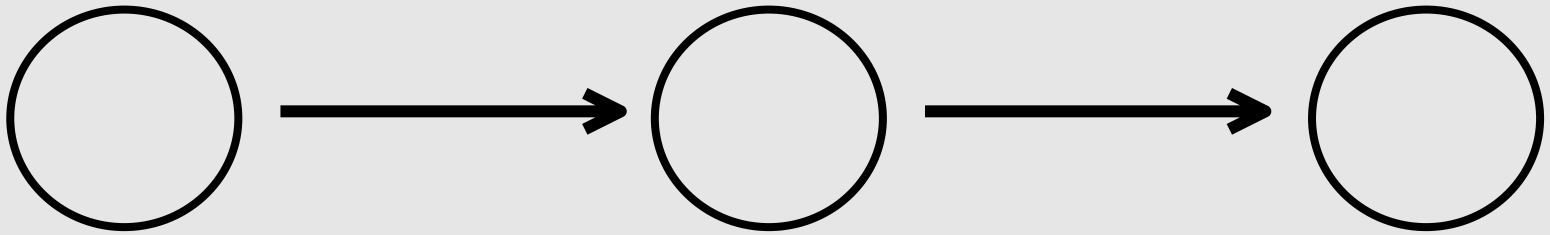
```
git_example on  main
 54% ❯ git log
commit 055635fafe1d67efb7d9e65f2bf558f6e5ec8841 (HEAD -> main)
Author: Ryan Schanzenbacher <ryan@rschanz.org>
Date:   Fri Sep 16 11:18:42 2022 -0400

    Bad commit!

commit 76c72fd01daf2fafb5ae8bb01de27ef0b18a8dcc
Author: Ryan Schanzenbacher <ryan@rschanz.org>
Date:   Fri Sep 16 11:08:06 2022 -0400

    Another commit

commit 99ac1f9c7e44f0bdba955571db5290749fa7f119
Author: Ryan Schanzenbacher <ryan@rschanz.org>
Date:   Fri Sep 16 09:46:43 2022 -0400

    My first commit
git_example on  main
 54% ❯ git revert 055635fafe...76c72fd01d
[main dd88f29] Revert "Bad commit!"
 1 file changed, 2 deletions(-)
git_example on  main took 8s
 55% ❯
```

What if.....

- I want to explore a previous state, but don’t want to actually commit it back?
  - Use `git checkout {commit_hash}`. That command will bring your working tree to the state of that commit, but you don’t erase any of your future commits. You can easily change back by running `git checkout {branch_name}`
- I want to restore a single file from a previous commit, not the entire repository
  - Use `git checkout {commit_hash} - - {path/to/file1 file2...}`
  - Make sure you add/commit after this to keep the revert!

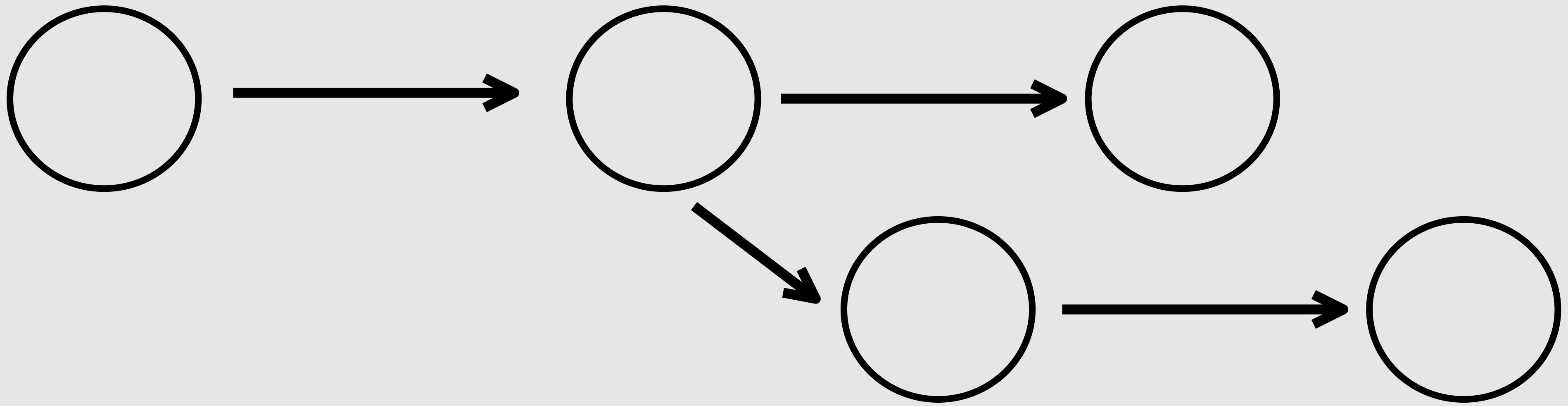
Branches!

- One of the most powerful concepts in git is the ability to branch your repository
- As of now, all of our commits have essentially looked like a line



Branches!

- However, what if you want to develop a new feature/test some code without wrecking your current state?
- This is where you use branches!
- This will make your commit history look something like this

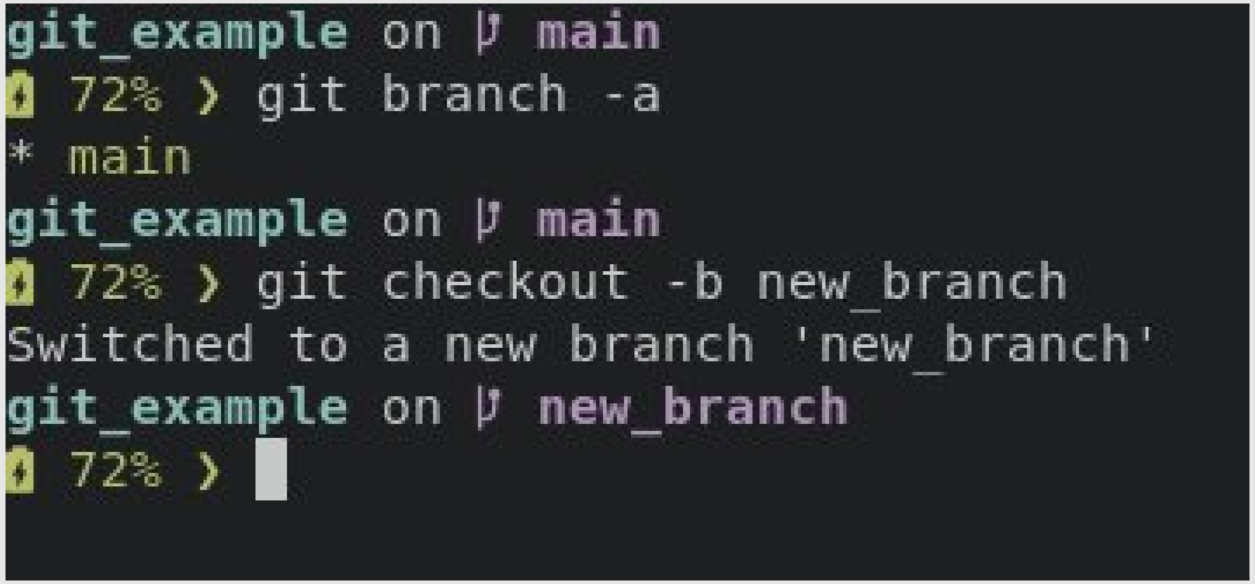


Branches!

- Their main purpose is to be a place for testing code to go, eventually to be merged back into the main branch
- Commits and development can happen on any number of branches concurrently
  - That is to say: You can switch between them and make commits to any one you want. This allows for non-linear development
- Another use they have is to maintain separate features for whatever reason may be needed
  - In that case, development just continues on each branch

How to use branches

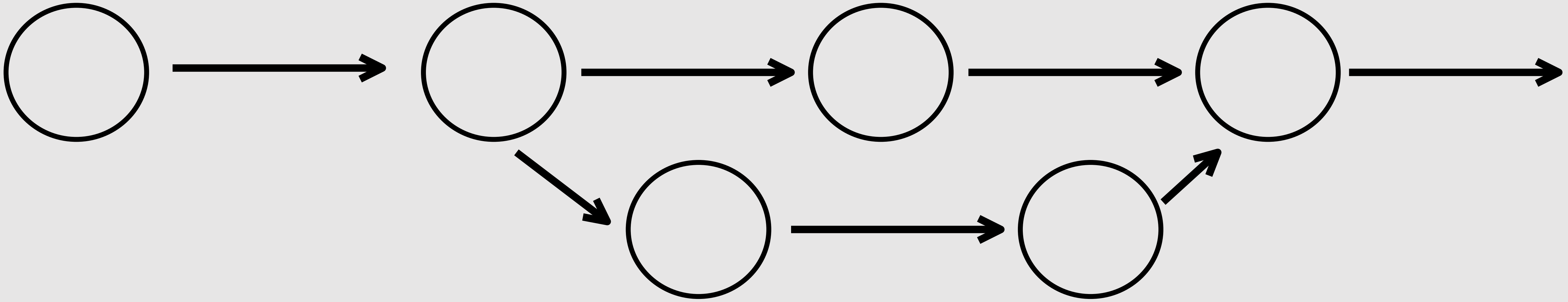
- First, let’s see what branches we have. `git branch –a` shows your current branches
- To create a new branch, use `git checkout –b {name}`

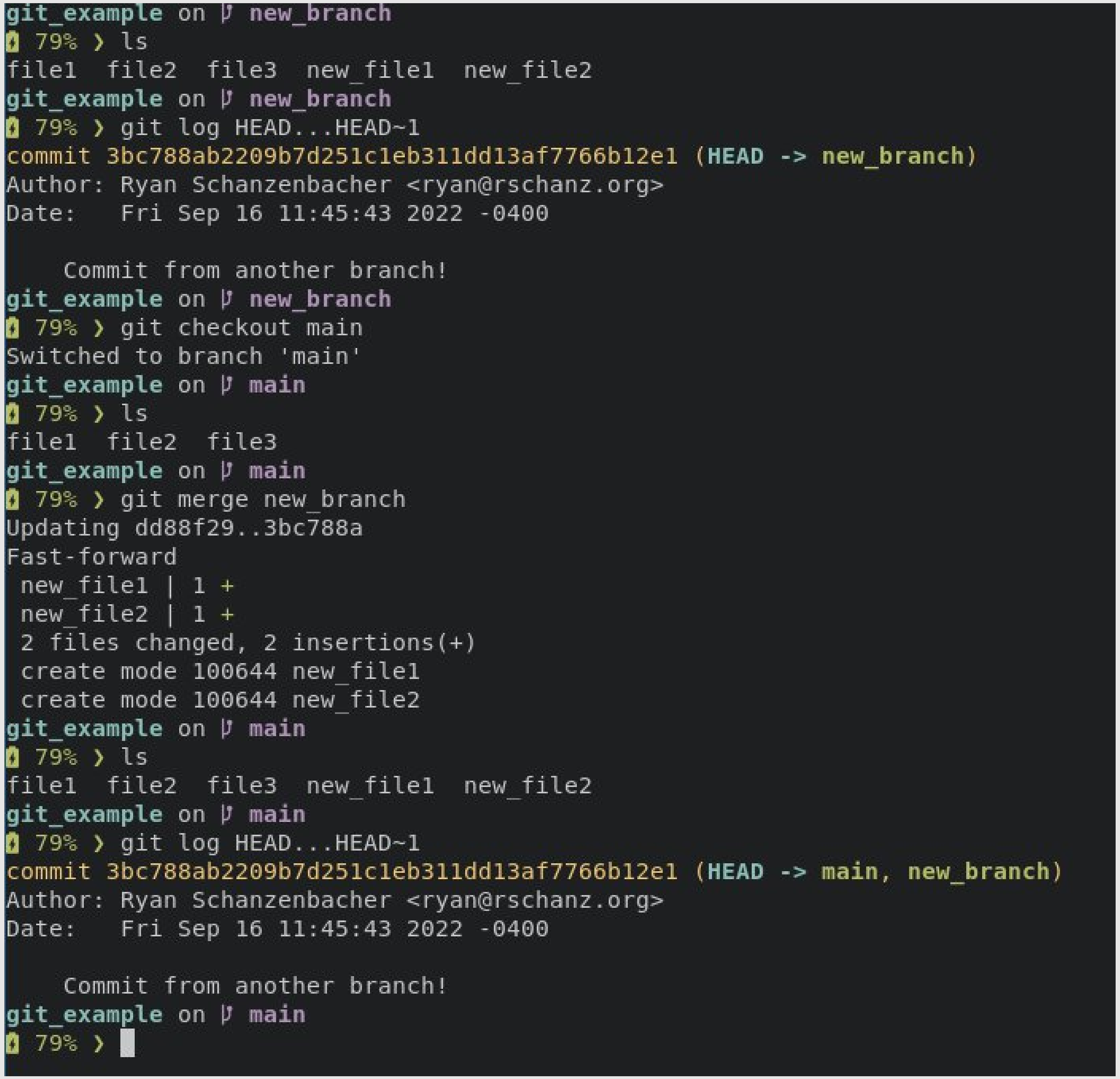


```
git_example on  main
 72% ❯ git branch -a
* main
git_example on  main
 72% ❯ git checkout -b new_branch
Switched to a new branch 'new_branch'
git_example on  new_branch
 72% ❯ 
```

How to use branches

- Now you just follow your workflow that we described before
- To switch between branches (make sure you commit first!) use the command `git checkout {branch_name}`
- When you're ready to merge your changes back into your main branch, you switch to the branch you want things to go into (usually main), then run the command `git merge {branch}`





```
git_example on  new_branch
 79% ❯ ls
file1  file2  file3  new_file1  new_file2
git_example on  new_branch
 79% ❯ git log HEAD...HEAD~1
commit 3bc788ab2209b7d251c1eb311dd13af7766b12e1 (HEAD -> new_branch)
Author: Ryan Schanzenbacher <ryan@rschanz.org>
Date:   Fri Sep 16 11:45:43 2022 -0400

    Commit from another branch!
git_example on  new_branch
 79% ❯ git checkout main
Switched to branch 'main'
git_example on  main
 79% ❯ ls
file1  file2  file3
git_example on  main
 79% ❯ git merge new_branch
Updating dd88f29..3bc788a
Fast-forward
 new_file1 | 1 +
 new_file2 | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 new_file1
 create mode 100644 new_file2
git_example on  main
 79% ❯ ls
file1  file2  file3  new_file1  new_file2
git_example on  main
 79% ❯ git log HEAD...HEAD~1
commit 3bc788ab2209b7d251c1eb311dd13af7766b12e1 (HEAD -> main, new_branch)
Author: Ryan Schanzenbacher <ryan@rschanz.org>
Date:   Fri Sep 16 11:45:43 2022 -0400

    Commit from another branch!
git_example on  main
 79% ❯ 
```

Github workflow!

- When working with Github, it is best practice to contribute to projects the following way:
  - Create a fork of the project (fork button in Github) and clone it to your machine (`git clone {URL}`)
  - When in the repo on your machine, create a new branch to develop your feature on
  - When you have all your commits made, push to your github fork
  - Github will prompt you (due to pushing a new branch to a fork) if you’d like to make a pull request
  - Draft the pull request and wait for feedback from maintainers!

Demo time!!

We’ve been looking at the porcelain... It’s time to look at the plumbing :)