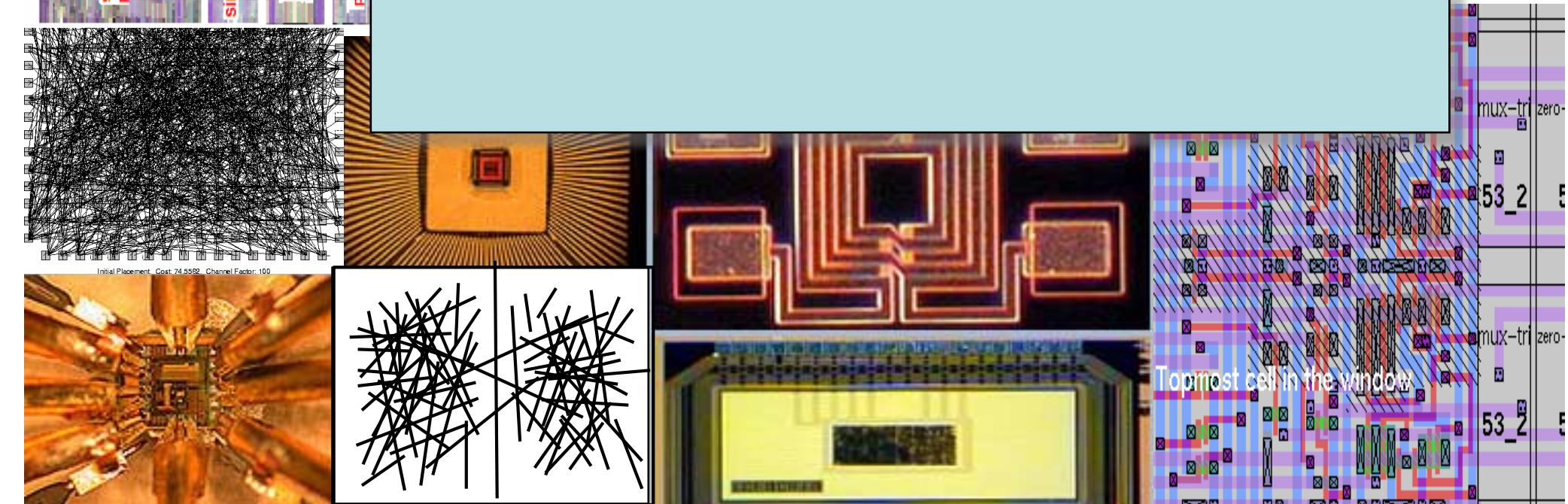


CPEN 391: Computer Systems Design Studio

Lecture 2: Revision Control



Managing Your Code

- The same thing...
 - Software Configuration Management
 - Revision Control
 - Version Control
- But what is it?
 - A practice for keeping track of changes to your source files
 - Tools to help you
- Who should do it?
 - Everyone!
 - Individual developers
 - Teams of developers
 - Hardware or software

Revision Control Systems – Tools

- Old guard
 - SCCS, RCS, DCS, PVCS, DDE, Perforce ...
- Microsoft
 - Visual SourceSafe (VSS) (deprecated)
 - Team Foundation Version Control (TFVC) or Git
- Open source favourites: CVS, Subversion, Mercurial
- The Rolls Royce: IBM Rational ClearCase(\$\$)
- New darling: GIT
 - Used by Linux kernel
 - Intended for wide-scale distributed development
 - Also useful for individuals and teams

RCS Tools – which to use?

- There are ~40 different RCS tools

- List

http://en.wikipedia.org/wiki/List_of_version-control_software

- Comparison

http://en.wikipedia.org/wiki/Comparison_of_revision_control_software

- This course: GIT
 - Most popular among CompEng profs in ECE
 - Used in my company
 - In particular, <http://github.com> is recommended
 - Free accounts for students
 - Free accounts for open source projects

RCS – simple revision control

RCS comes with Linux and (all) versions of Unix

- Often used to remember changes to “Linux config files”

- Save a copy of the current file, as a revision:

```
$ ci dhcpd.conf
```

- This is stored in file named dhcpd.conf,v

- Check out a read-only copy of the file for use:

```
$ co dhcpd.conf
```

- Check out a writable copy, or make read-only copy writable:

```
$ co -l dhcpd.conf      (dash-lowercase-L for “lock”)
```

(modify file)

```
$ ci dhcpd.conf          (save new version)
```

GIT – Basics

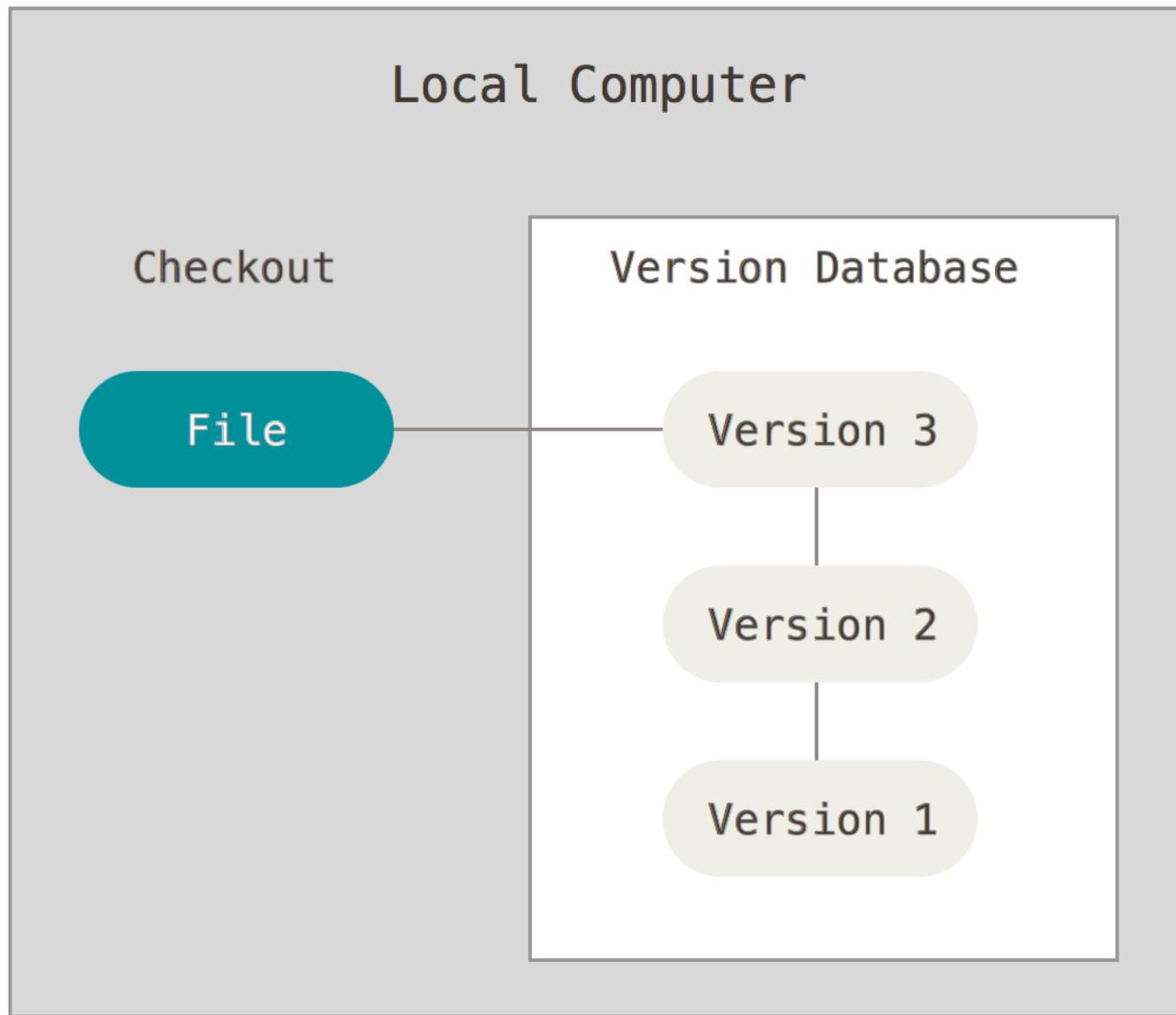
- Command-line interface (standardized)
 - Many GUI tools, not really standardized
- Use under “Linux” or “Cygwin on Windows”
- Eg, install it as part of Cygwin:
 - Go to cygwin.com, download setup-x86.exe or setup-x86_64.exe
 - Start → Altera → Nios → Nios Command Shell
 - cd /cygdrive/c/users/myusername/downloads
 - ./setup-x86_64.exe
 - Install under c:\cygwin64
 - Select git, git-completion, git-gui, gitk packages
 - Install them (and any dependencies)

Git

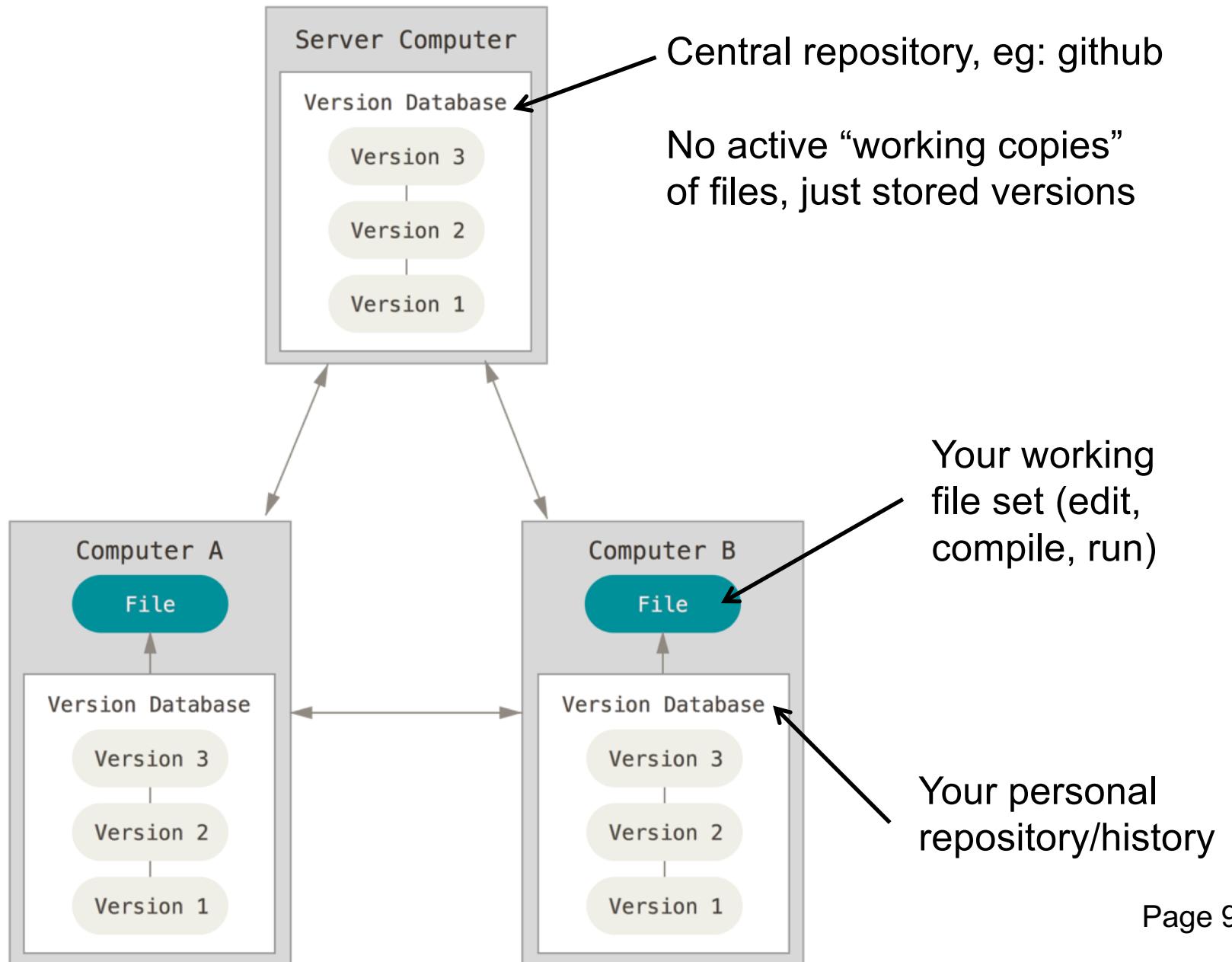
- Git works with “repositories”
 - It is decentralized by nature
 - But you can use a central repository (best for small teams)
- Great GIT tutorials
 - <http://www.atlassian.com/git/tutorials>
 - Eg: “Using Branches” and “Comparing Workflows”
 - Aside: Atlassian runs BitBucket, which is similar to GitHub
- Official documentation
 - <http://git-scm.com/doc>

This lecture will borrow figures and content from both of these sites, and more ...

Git – Main Idea



Git – Decentralized Control



Creating Git Repositories

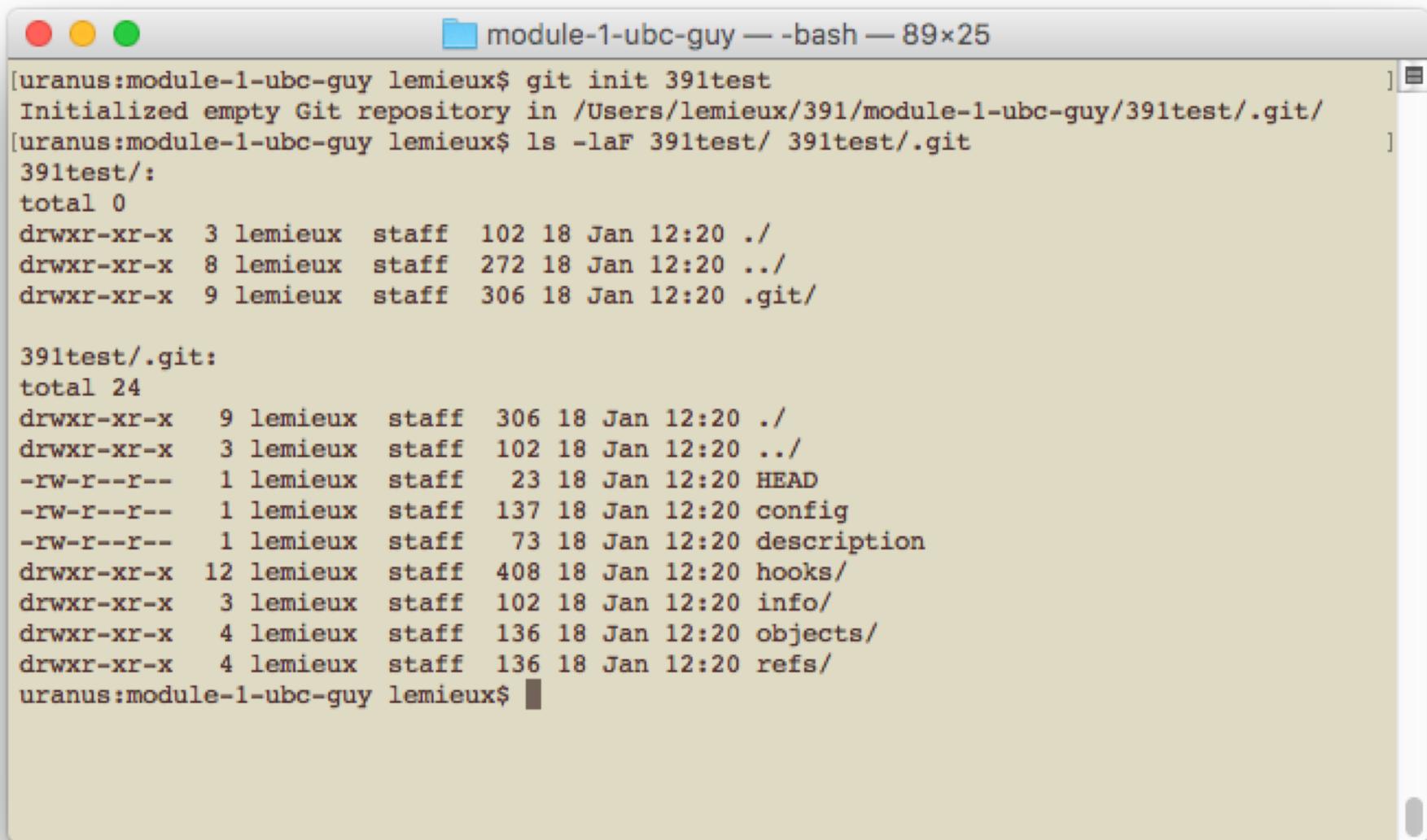
Cover 2 ways to create git repositories

1. Local repository (on PC, personal development only)
2. Remote repository (in cloud, personal or team-based)

Git – Getting Started (Create Local Repository)

- Private local repository on your computer:

```
$ git init 391test
```



The screenshot shows a terminal window titled "module-1-ubc-guy — -bash — 89x25". The window contains the following text output from a terminal session:

```
[uranus:module-1-ubc-guy lemieux$ git init 391test
Initialized empty Git repository in /Users/lemieux/391/module-1-ubc-guy/391test/.git/
[uranus:module-1-ubc-guy lemieux$ ls -laF 391test/ 391test/.git
391test/:
total 0
drwxr-xr-x  3 lemieux  staff  102 18 Jan 12:20 .
drwxr-xr-x  8 lemieux  staff  272 18 Jan 12:20 ../
drwxr-xr-x  9 lemieux  staff  306 18 Jan 12:20 .git/
391test/.git:
total 24
drwxr-xr-x  9 lemieux  staff  306 18 Jan 12:20 .
drwxr-xr-x  3 lemieux  staff  102 18 Jan 12:20 ../
-rw-r--r--  1 lemieux  staff   23 18 Jan 12:20 HEAD
-rw-r--r--  1 lemieux  staff  137 18 Jan 12:20 config
-rw-r--r--  1 lemieux  staff   73 18 Jan 12:20 description
drwxr-xr-x 12 lemieux  staff  408 18 Jan 12:20 hooks/
drwxr-xr-x  3 lemieux  staff  102 18 Jan 12:20 info/
drwxr-xr-x  4 lemieux  staff  136 18 Jan 12:20 objects/
drwxr-xr-x  4 lemieux  staff  136 18 Jan 12:20 refs/
uranus:module-1-ubc-guy lemieux$ ]
```

Git – Getting Started (Create Remote Repository)

The screenshot shows the GitHub 'Create a New Repository' interface. On the left, there's a sidebar with 'Repository template' (set to 'No template'), 'Owner' (set to 'ubc-guy'), and 'Repository name' ('391test'). The main area has fields for 'Description (optional)' and 'Initialize this repository with:' (checkboxes for 'Add a README file', 'Add .gitignore', and 'Choose a license', all checked). A purple arrow points from a callout box to the 'Private' radio button under 'Initialize this repository with:', which is currently unselected (unchecked). The callout box contains the text: 'For coursework, make this private!'. At the bottom is a green 'Create repository' button.

For coursework,
make this private!

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner * Repository name *

ubc-guy / 391test ✓

Great repository names are short and memorable. Need inspiration? How about cu

Description (optional)

Description (optional)

Description (optional)

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. [Learn more](#).

Add .gitignore
Choose which files not to track from a list of templates. [Learn more](#).

Choose a license
A license tells others what they can and can't do with your code. [Learn more](#).

.gitignore template: C ▾

License: Apache License 2.0 ▾

This will set `main` as the default branch. Change the default name in your [settings](#).

Create repository

Git – Getting Started (Remote Repository Result)

The screenshot shows a GitHub repository page for the user 'ubc-guy' named '391test'. The repository is public and contains one branch ('main') and three files: '.gitignore', 'LICENSE', and 'README.md'. The 'README.md' file content is '391test'. The repository has 1 commit from 'ubc-guy' made 1 minute ago. The commit message is 'Initial commit'. The repository has 0 stars, 1 watching, and 0 forks. There are no releases or packages published.

ubc-guy / 391test Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

ubc-guy Initial commit a4f5113 1 minute ago 1 commit

.gitignore Initial commit 1 minute ago

LICENSE Initial commit 1 minute ago

README.md Initial commit 1 minute ago

README.md

391test

About

No description, website, or topics provided.

Readme 0 stars 1 watching 0 forks

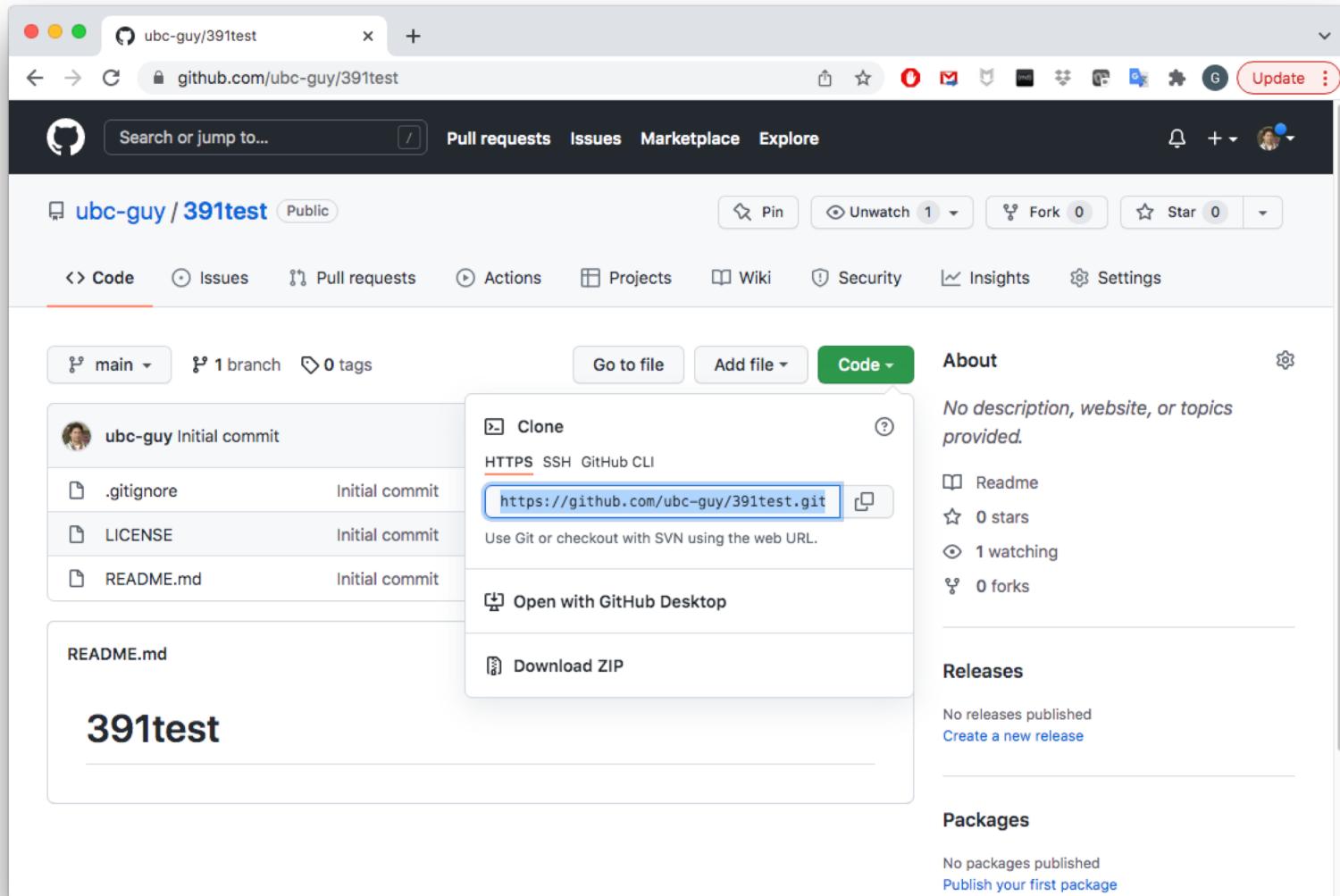
Releases

No releases published Create a new release

Packages

No packages published Publish your first package

Git – Getting Started (URL to Remote Repository)



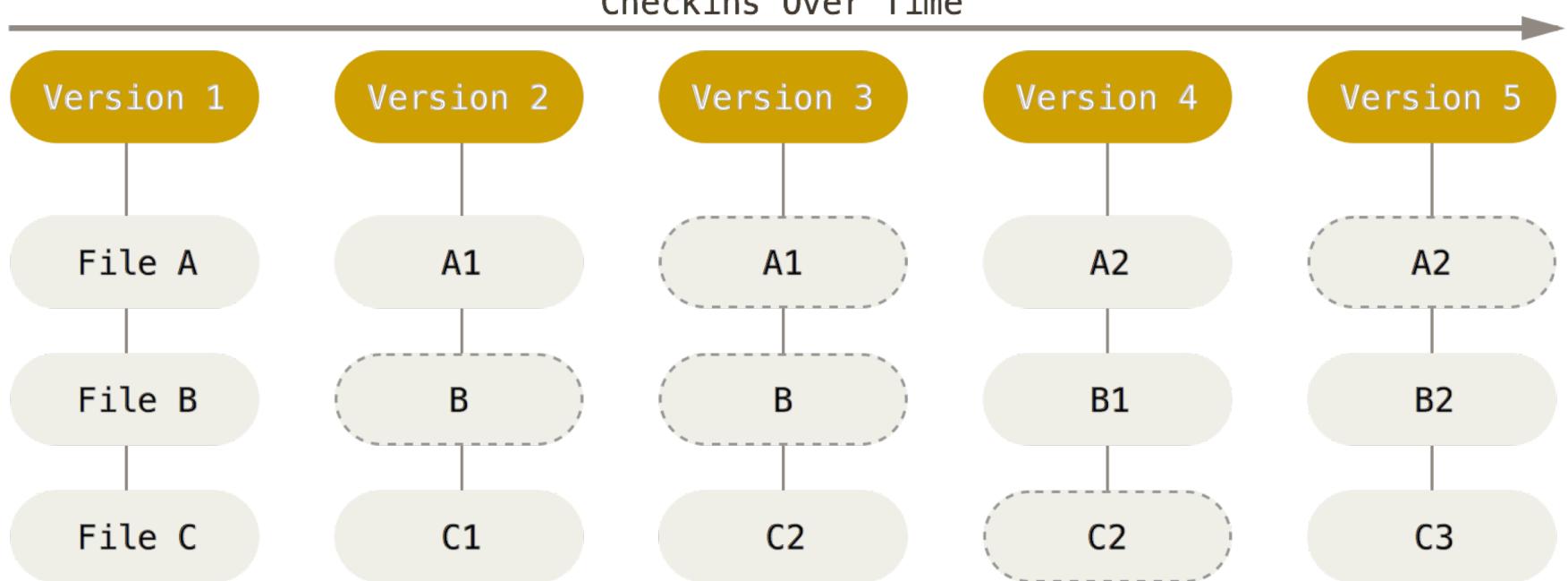
Git – Getting Started (Local Clone of Remote Repository)

```
[uranus:module-1-ubc-guy lemieux$ rm -rf 391test/
[uranus:module-1-ubc-guy lemieux$ git clone https://github.com/ubc-guy/391test.git
Cloning into '391test'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
[uranus:module-1-ubc-guy lemieux$ ls -lAF 391test/ 391test/.git
391test/:
total 40
drwxr-xr-x 12 lemieux staff 408 18 Jan 12:23 .git/
-rw-r--r-- 1 lemieux staff 430 18 Jan 12:23 .gitignore
-rw-r--r-- 1 lemieux staff 11357 18 Jan 12:23 LICENSE
-rw-r--r-- 1 lemieux staff 9 18 Jan 12:23 README.md

391test/.git:
total 40
-rw-r--r-- 1 lemieux staff 21 18 Jan 12:23 HEAD
-rw-r--r-- 1 lemieux staff 304 18 Jan 12:23 config
-rw-r--r-- 1 lemieux staff 73 18 Jan 12:23 description
drwxr-xr-x 12 lemieux staff 408 18 Jan 12:23 hooks/
-rw-r--r-- 1 lemieux staff 289 18 Jan 12:23 index
drwxr-xr-x 3 lemieux staff 102 18 Jan 12:23 info/
drwxr-xr-x 4 lemieux staff 136 18 Jan 12:23 logs/
drwxr-xr-x 9 lemieux staff 306 18 Jan 12:23 objects/
-rw-r--r-- 1 lemieux staff 105 18 Jan 12:23 packed-refs
drwxr-xr-x 5 lemieux staff 170 18 Jan 12:23 refs/
uranus:module-1-ubc-guy lemieux$
```

Git Files – Checking In

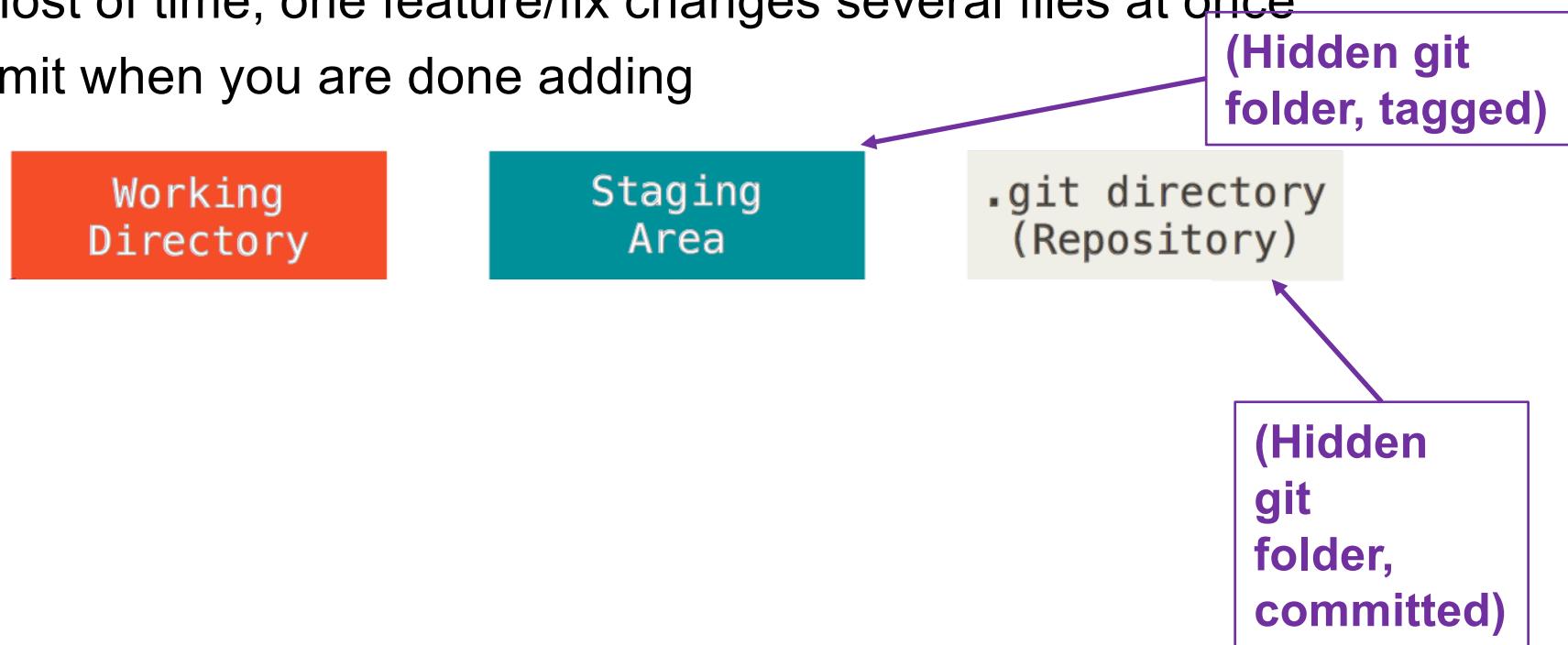
- Git stores snapshots of your file set, saved over time



- Nearly every operation is local (does not involve a remote server)
- Every version is stored and named using a SHA-1 hash, eg:
24b9da6552252987aa493b52f8696cd6d3b00373
- Git usually only adds data
 - Don't panic, almost any operation can be “undone”

Git – Staging Area

- Two stages: “add” followed by “commit”
- Add allows you to build one consistent set of files into staging area
 - Most of time, one feature/fix changes several files at once
- Commit when you are done adding



Basic Git Workflow

The basic Git workflow goes something like this:

1. You modify files in your working directory.

(modify file1.c, file2.c, file.h)

2. You stage the files, using “git add”, which puts a snapshot of each added file in the staging area.

\$ git add file1.c file2.c file.h

(modify primes.c)

\$ git add primes.c

3. You do a commit, using “git commit” which takes the files from the staging area and stores that permanently in your Git repository

\$ git commit

Git – Your first commit



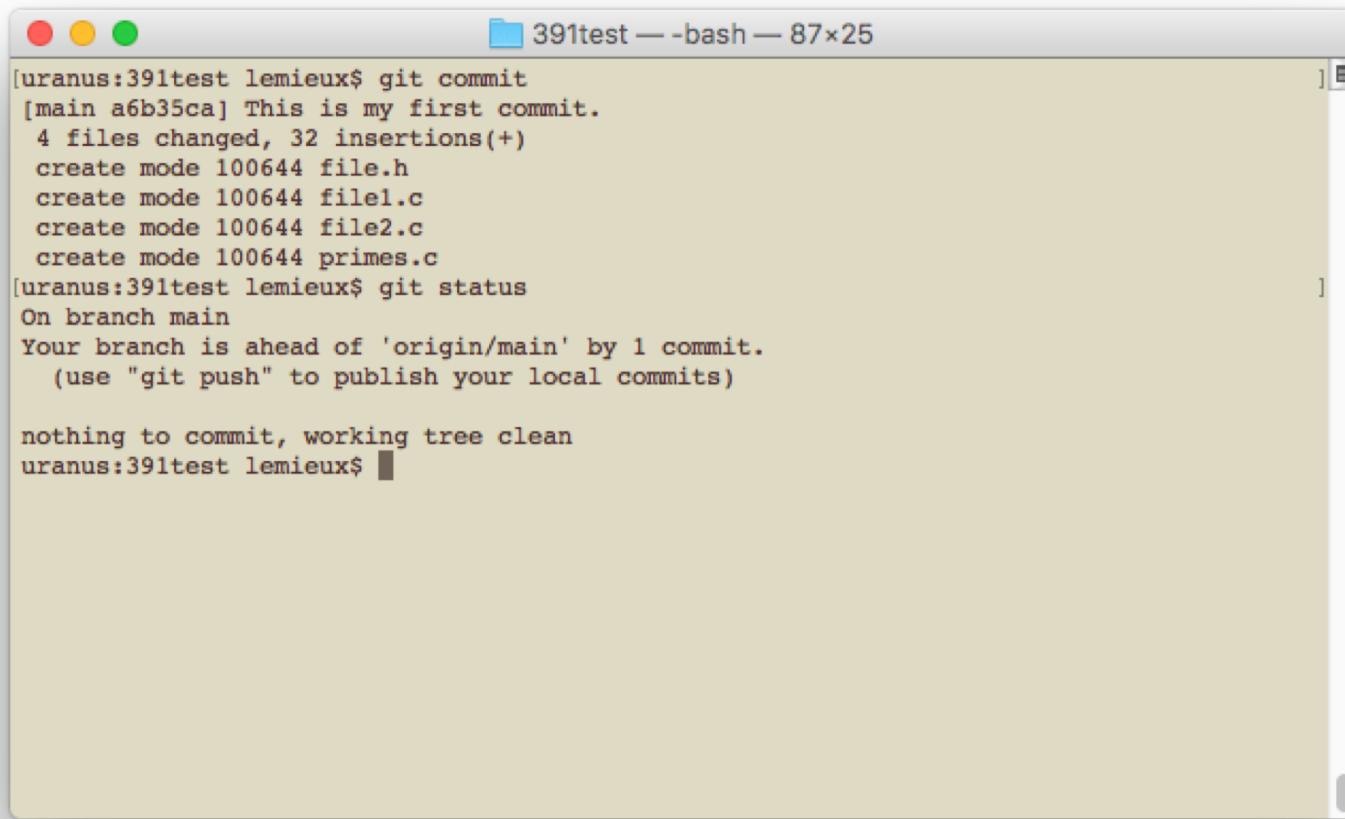
A screenshot of a macOS terminal window. The window title is "391test — -bash — 87x25". The terminal prompt shows "uranus:391test lemieux\$ git commit". The terminal window has a light beige background and is surrounded by a dark gray border. The window has red, yellow, and green close buttons in the top-left corner.

Git Commit – interactive editor

Git Commit – interactive editor

Git Commit – status after commit

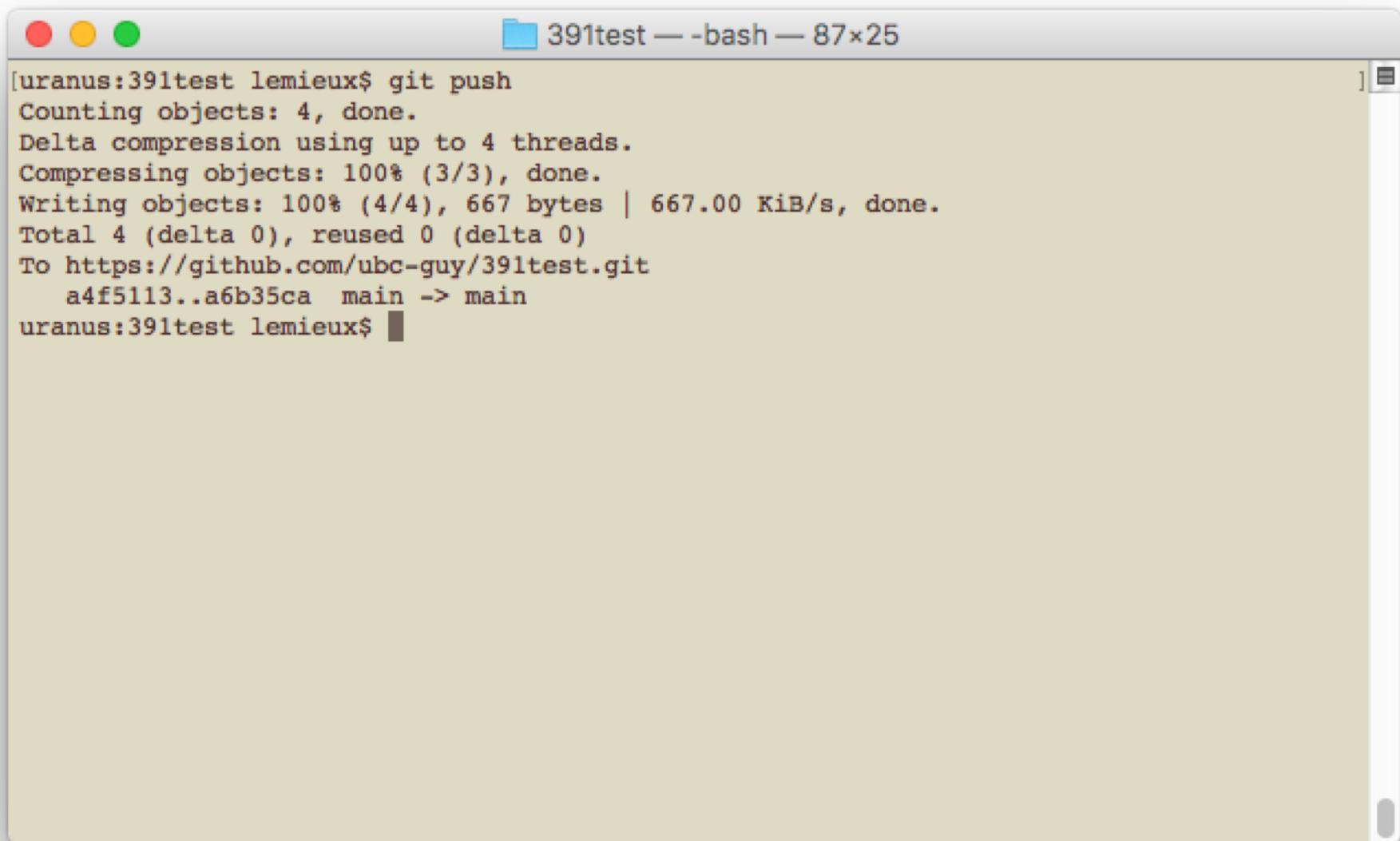
- Your changes are in your local PC repository
 - They have not been sent to the “shared repository”
- At any time, you can run “git status” to receive a short report on what has changed from the repository:



```
[uranus:391test lemieux$ git commit
[main a6b35ca] This is my first commit.
 4 files changed, 32 insertions(+)
 create mode 100644 file.h
 create mode 100644 file1.c
 create mode 100644 file2.c
 create mode 100644 primes.c
[uranus:391test lemieux$ 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
uranus:391test lemieux$ ]
```

Git – “push” to central repository



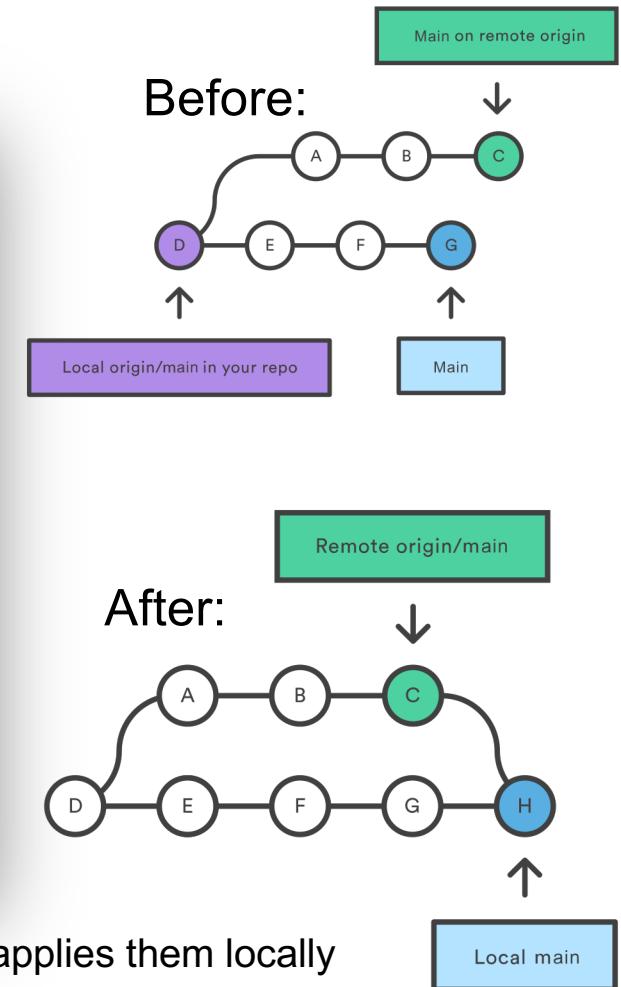
The terminal window shows the following command and its execution:

```
[uranus:391test lemieux$ git push
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 667 bytes | 667.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/ubc-guy/391test.git
  a4f5113..a6b35ca main -> main
uranus:391test lemieux$ ]
```

Git – Collaborations

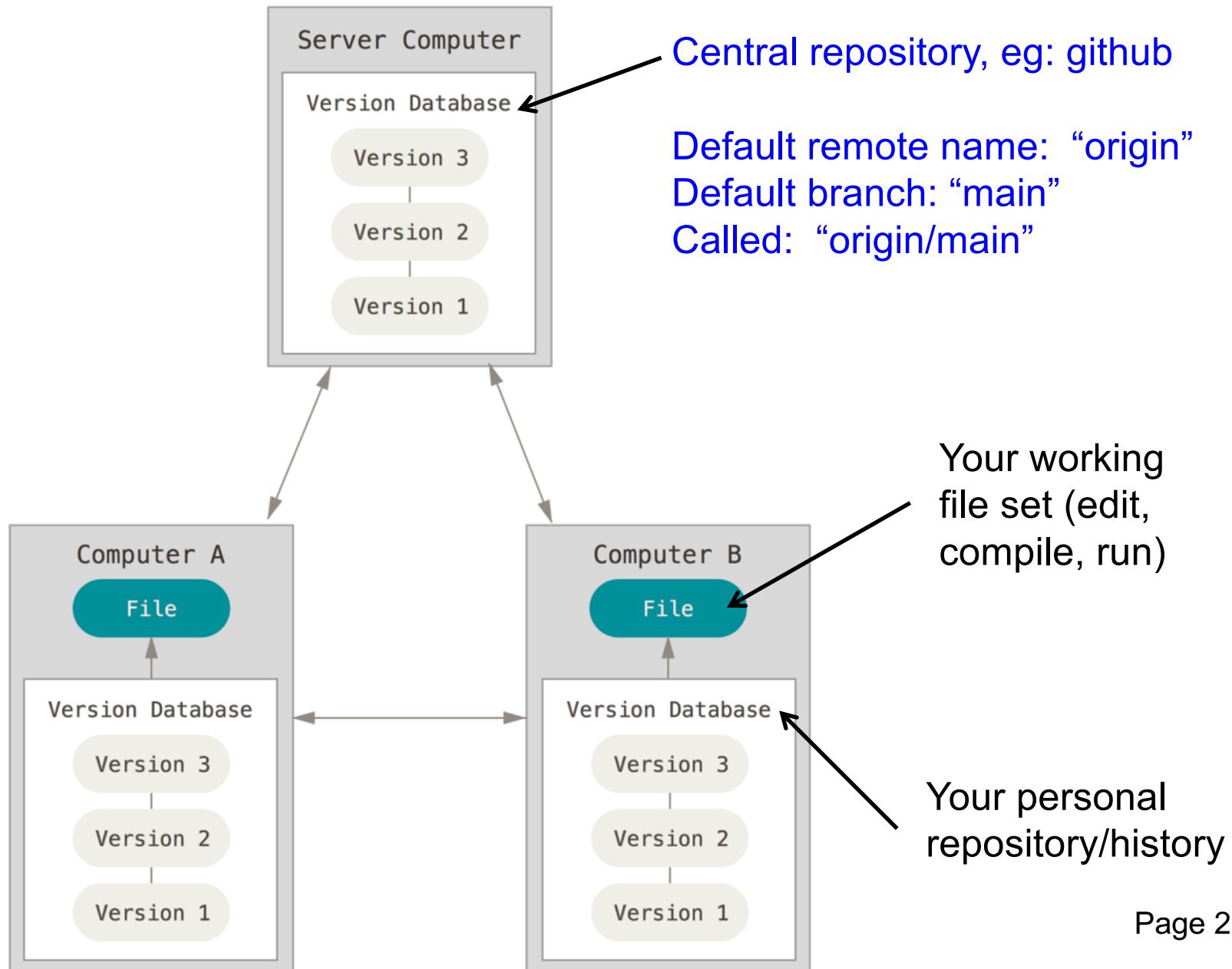
- Suppose my team-mate has modified primes.c and file.h
 - They correctly did “git add”, “git commit”, and “git push”
 - Retrieve all changes using “git pull”

```
uranus:391test lemieux$ git pull
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 3), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/ubc-guy/391test
  a6b35ca..29340d2  main      -> origin/main
Updating a6b35ca..29340d2
Fast-forward
 file.h | 1 +
 primes.c | 2 ++
 2 files changed, 3 insertions(+)
uranus:391test lemieux$
```

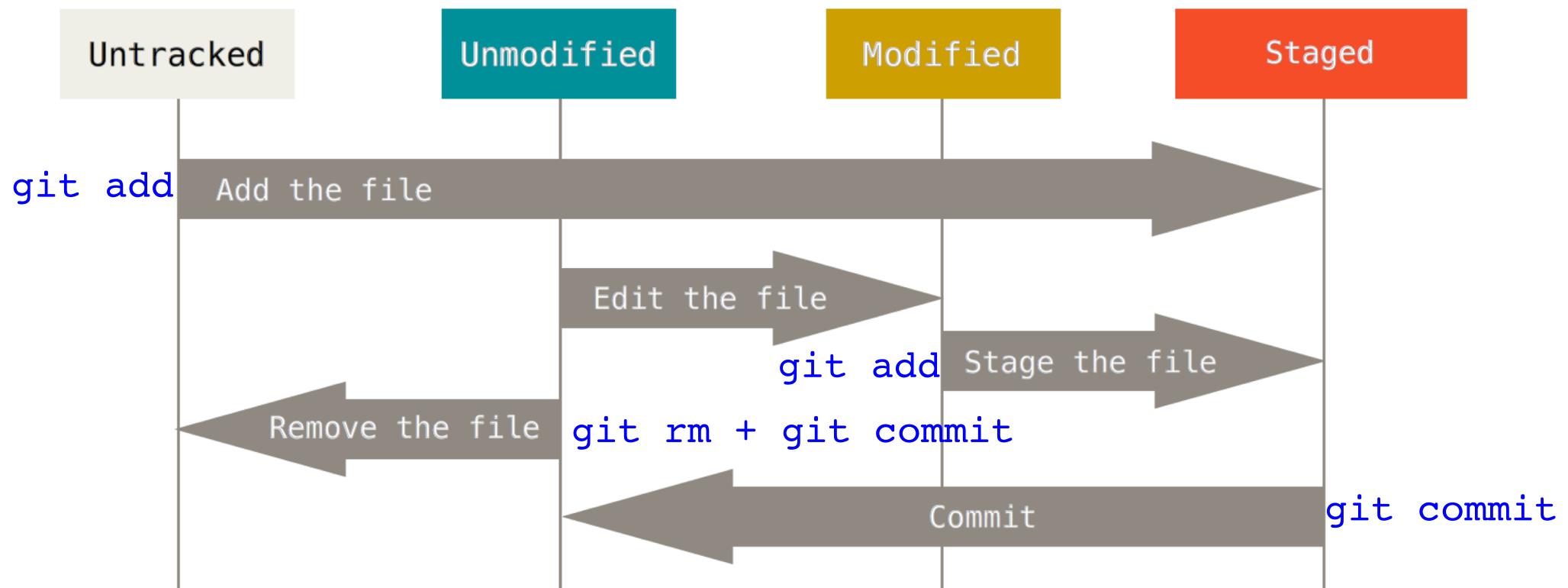


- Note: “git pull” retrieves changes to shared repository and applies them locally
- If I had also modified primes.c, there would be a conflict – need to merge results

Git – Decentralized Control

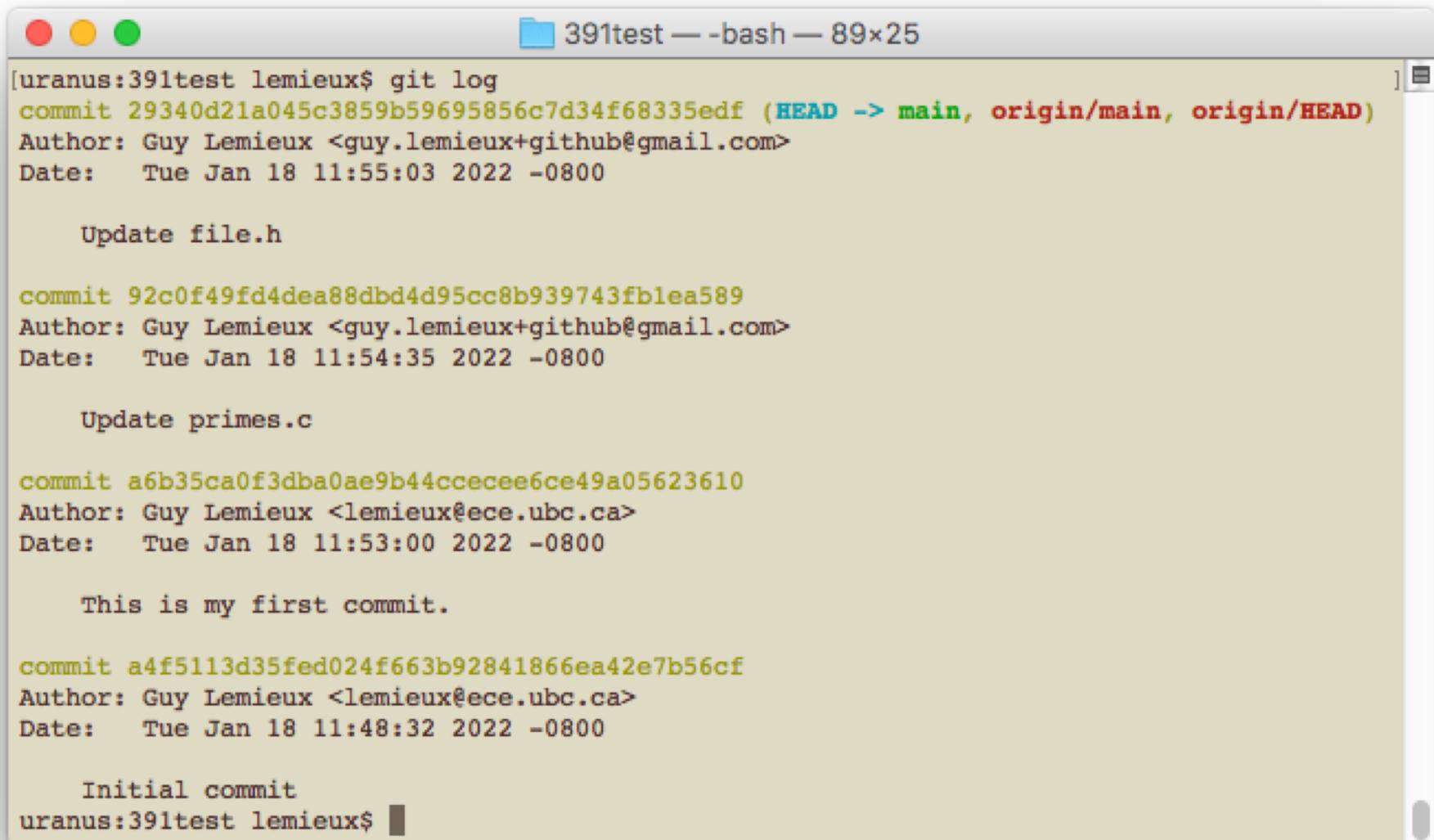


Git – File Life Cycle



Also: `git mv oldname.c newname.c`
`git commit`

Git – Viewing History



A screenshot of a terminal window titled "391test — -bash — 89x25". The window shows the output of the command "git log". The log output is as follows:

```
[uranus:391test lemieux$ git log
commit 29340d21a045c3859b59695856c7d34f68335edf (HEAD -> main, origin/main, origin/HEAD)
Author: Guy Lemieux <guy.lemieux+github@gmail.com>
Date:   Tue Jan 18 11:55:03 2022 -0800

    Update file.h

commit 92c0f49fd4dea88dbd4d95cc8b939743fb1ea589
Author: Guy Lemieux <guy.lemieux+github@gmail.com>
Date:   Tue Jan 18 11:54:35 2022 -0800

    Update primes.c

commit a6b35ca0f3dba0ae9b44ccece6ce49a05623610
Author: Guy Lemieux <lemieux@ece.ubc.ca>
Date:   Tue Jan 18 11:53:00 2022 -0800

    This is my first commit.

commit a4f5113d35fed024f663b92841866ea42e7b56cf
Author: Guy Lemieux <lemieux@ece.ubc.ca>
Date:   Tue Jan 18 11:48:32 2022 -0800

    Initial commit
uranus:391test lemieux$ ]
```

Git – Summary So Far

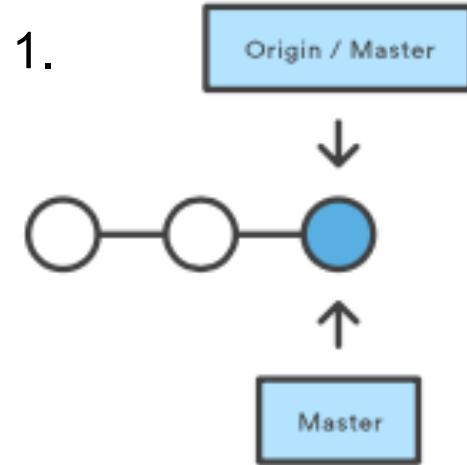
```
$ git clone https://github.com/name/project.git
$ cd project
(create/modify files)
$ git add file1.c file2.c file.h
(create/modify primes.c)
$ git add primes.c
$ git commit
$ git push
$ git status
```

<...somebody else works on files...>

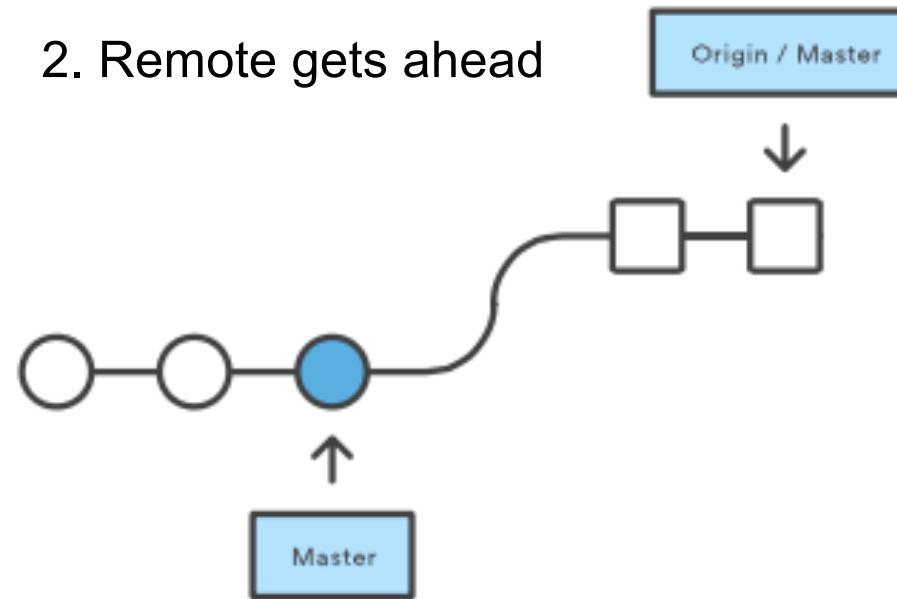
```
$ git pull
```

Git Pull – moving heads

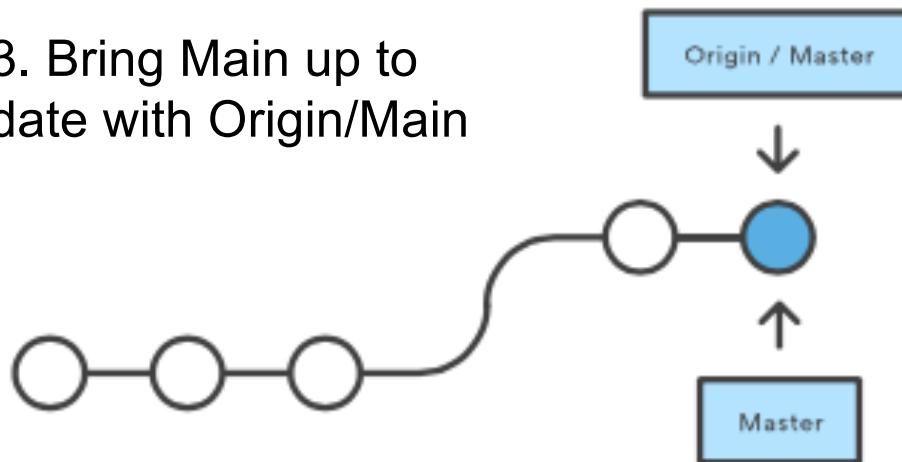
- Git pull



2. Remote gets ahead



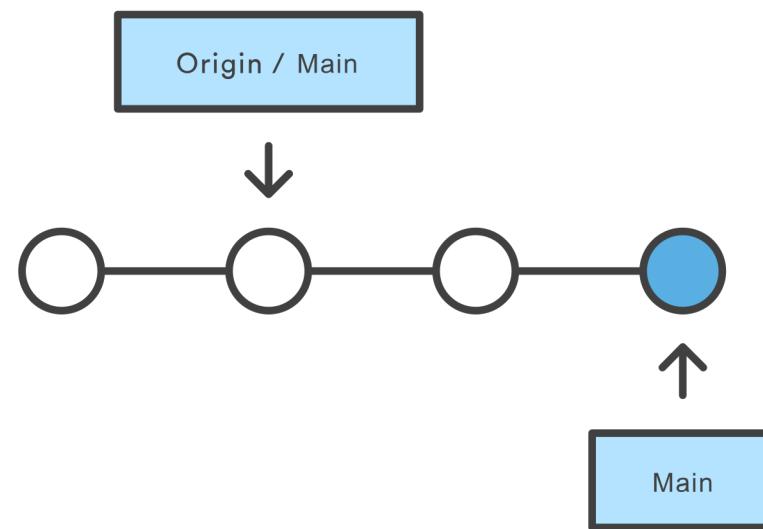
3. Bring Main up to date with Origin/Main



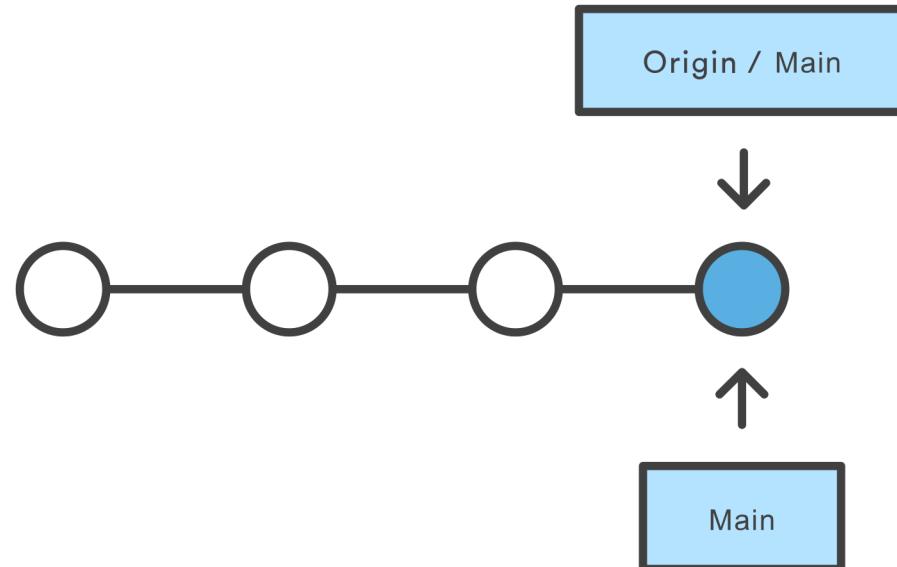
Repositories created before Oct 2020 used “Master”.
Now, they use “Main”.

• Git Push – moving heads

- Before pushing



- After pushing



Git – Throwing away changes

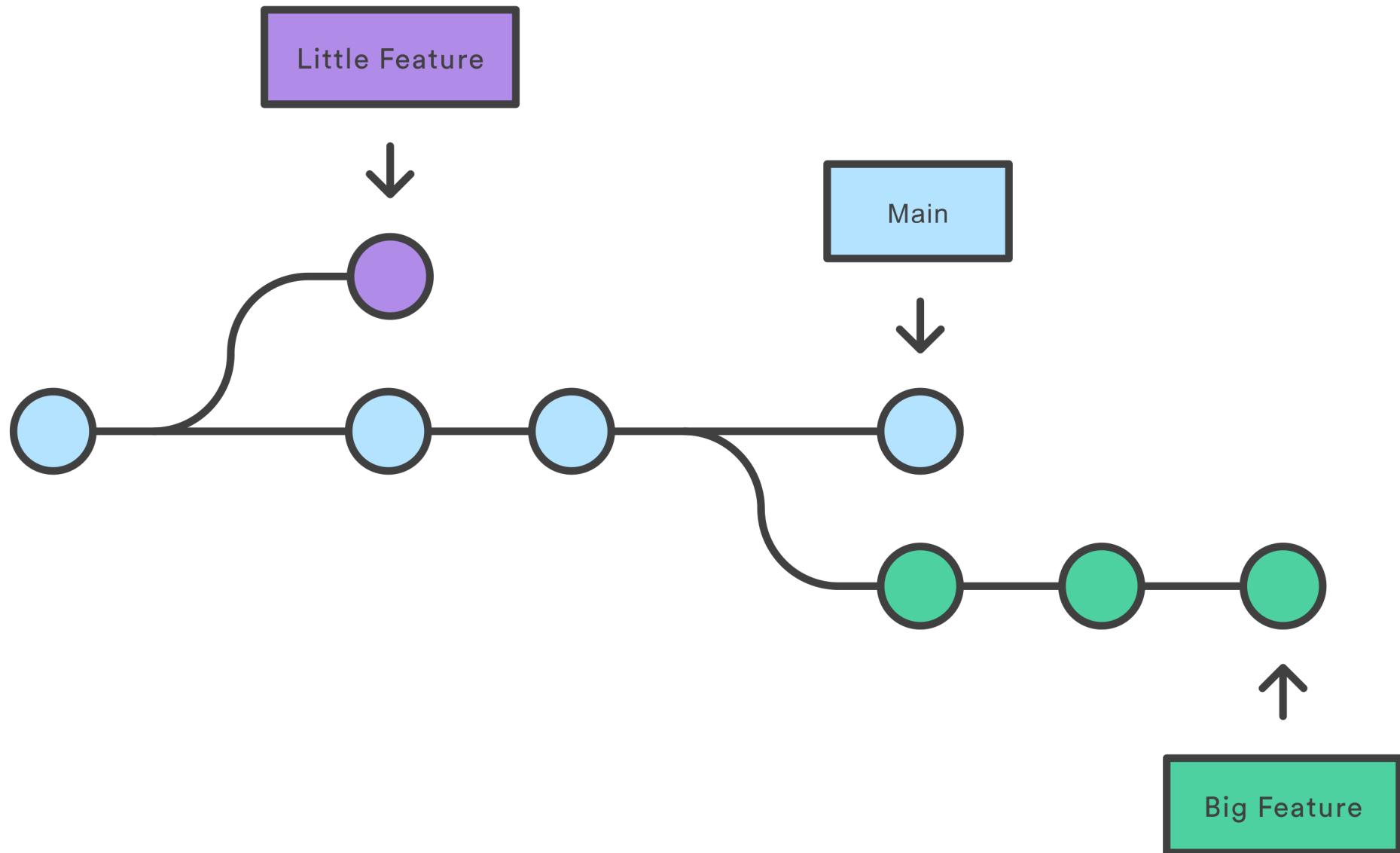
- You started to modify file1.c
 - Decided your approach was wrong
 - Want to start over with the last committed (staged) file1.c
 - Even if you (accidentally) removed file1.c
 - Restore it with the version from local repository

```
$ rm file1.c  
$ git checkout file1.c
```

Git – Branches

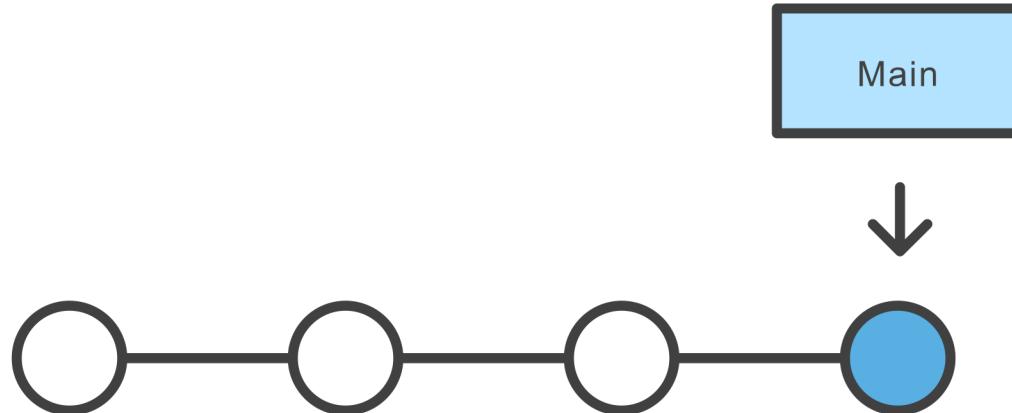
- By default, you are working on “Main” branch
 - You should never do this!
- Instead
 - Create new branches
 - Merge them back into Main
- Need a workflow

Git – Branches



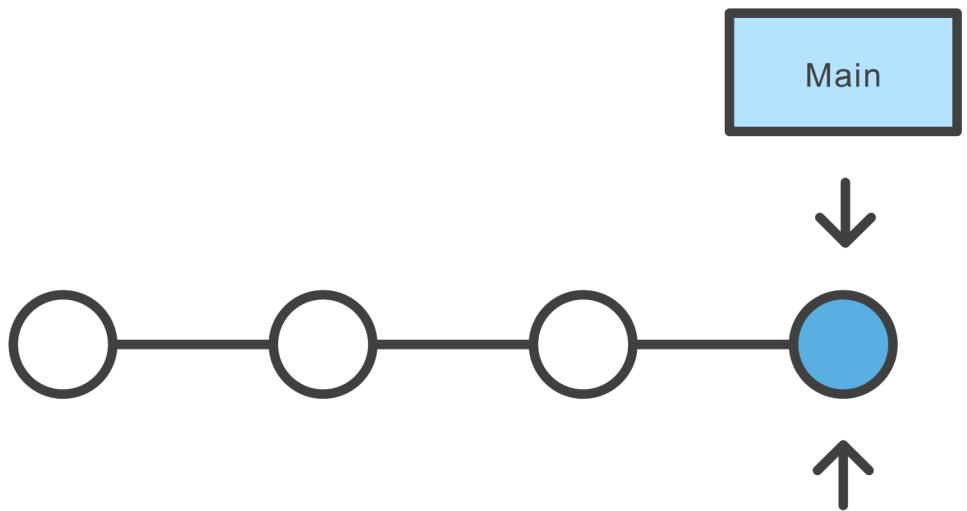
Git – Creating Branches

Before:



\$ git branch crazy-experiment (creates branch in repo)

After:

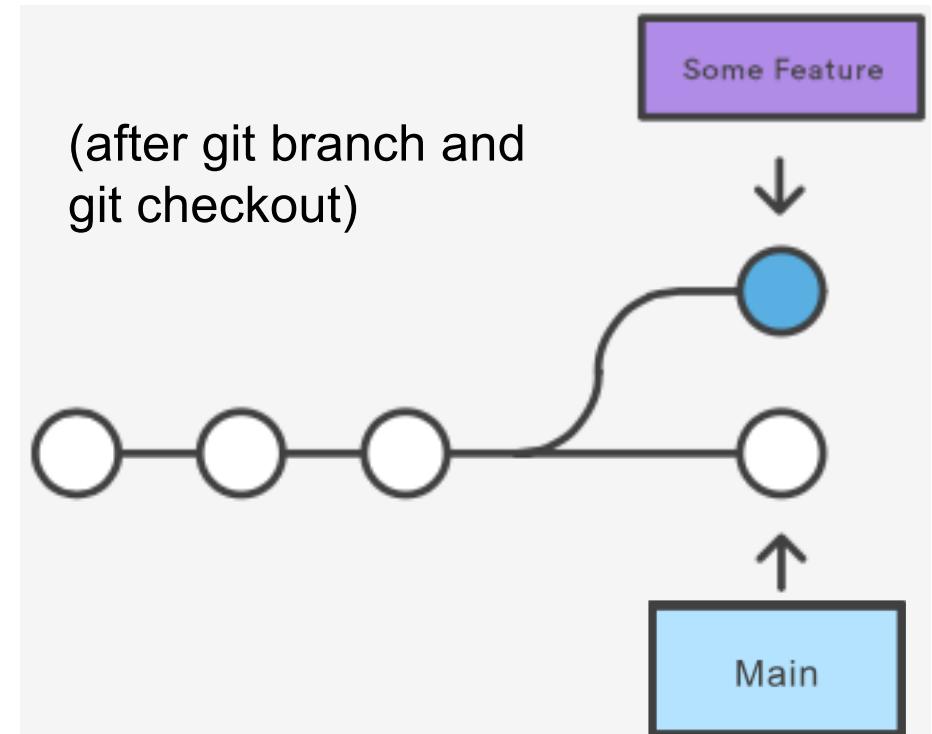
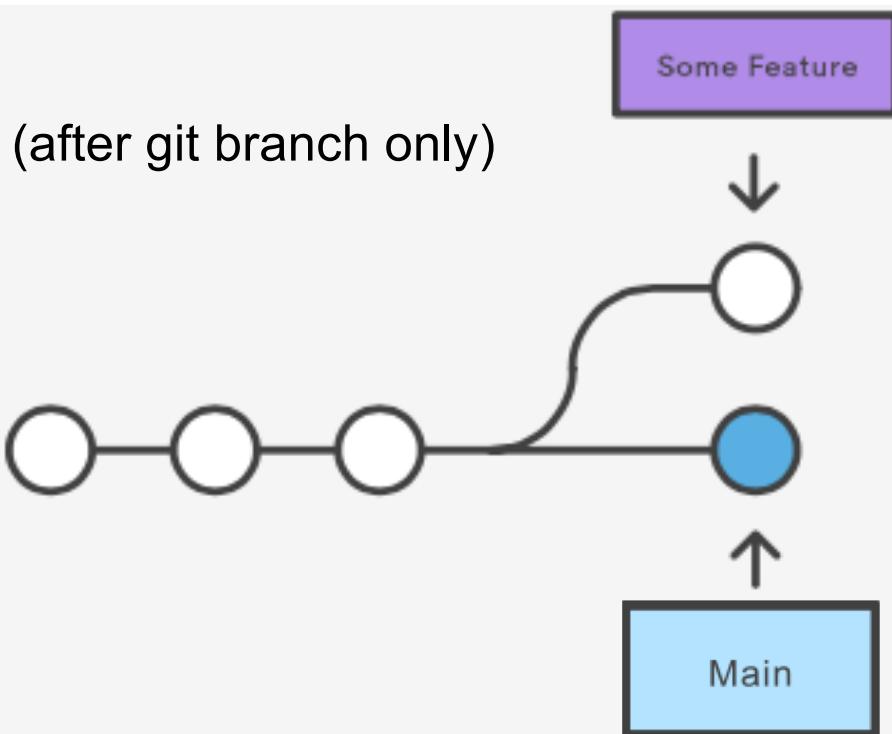


Git – Checking Out a Branch

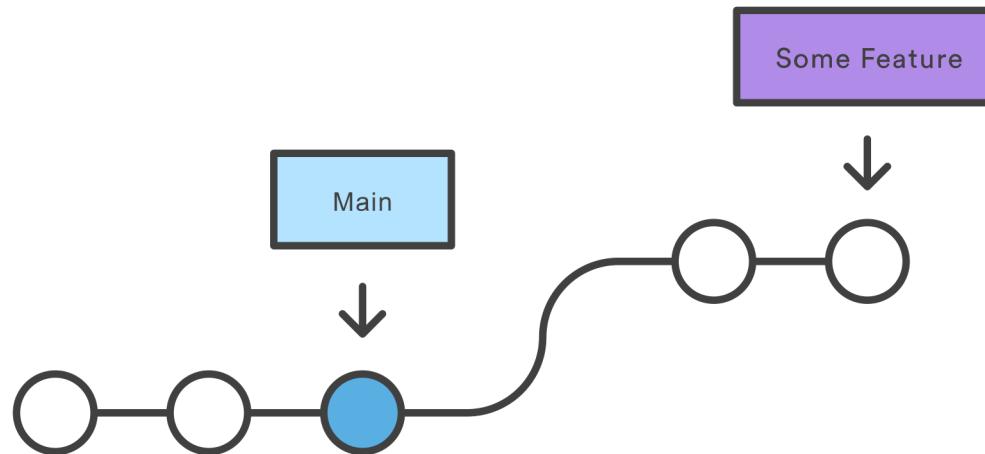
```
$ git branch some-feature          (creates branch in repo)  
$ git checkout some-feature        (selects branch for commits)  
(or)  
$ git checkout -b some-feature    (does both in one cmd)
```

To send this new local branch to the remote repository:

```
$ git push -u origin some-feature
```



Git – Merge

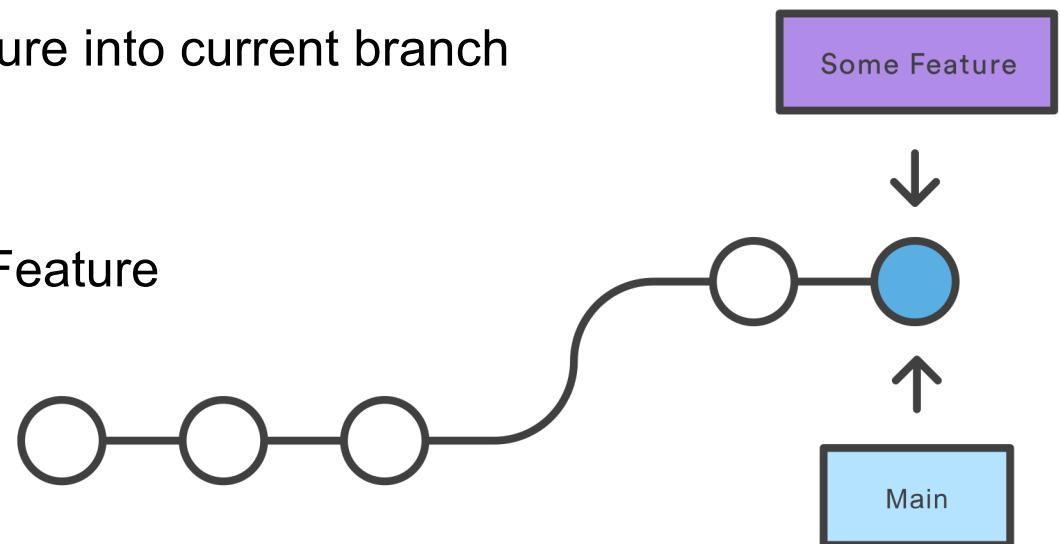


```
$ git checkout main          (select Main branch for commits)  
$ git merge some-feature (fast-forward Main)
```

Merge brings contents of some-feature into current branch

Simple case:

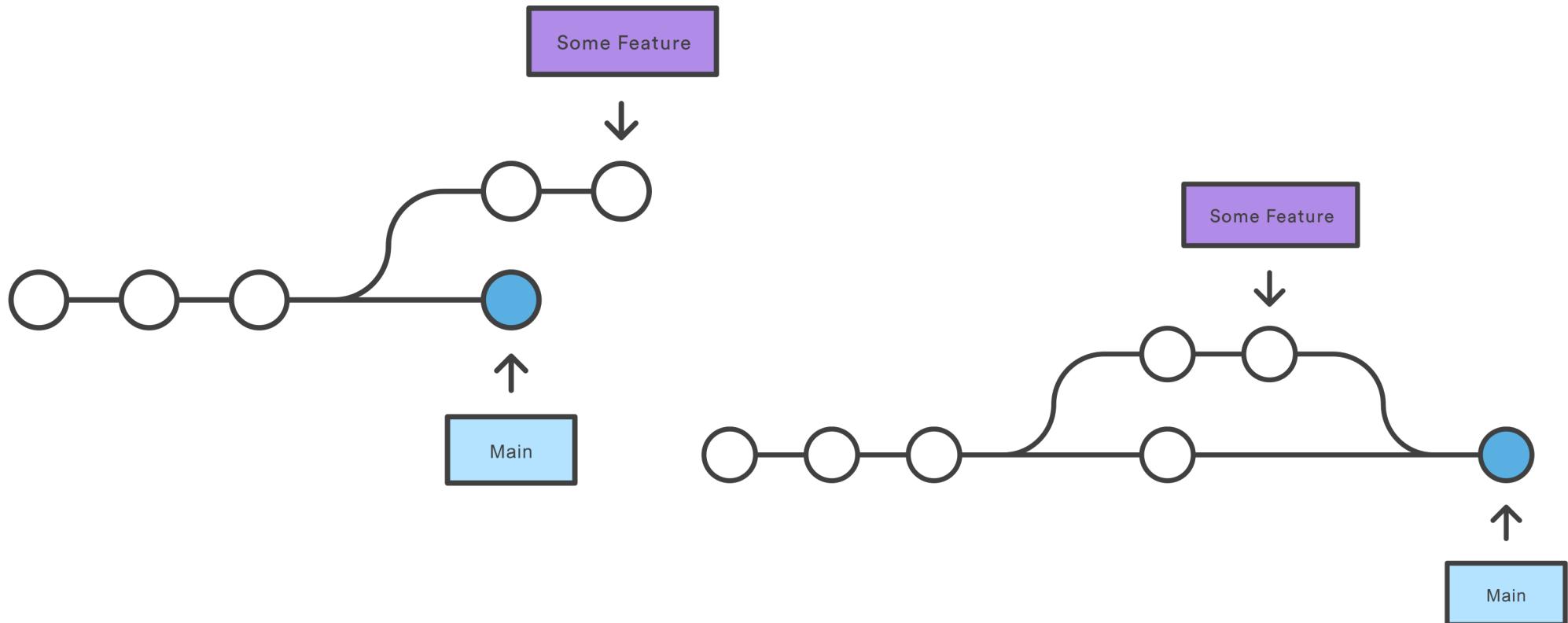
- Linear path from Main to Some Feature
- Called a fast-forward merge



Git – Merge

```
$ git checkout main    (select Main branch for commits)  
$ git merge some-feature
```

- No linear path from Main to some-feature, called 3-way merge
 - Git uses 3 commits to complete (Main, some-feature, then merged Main)
 - Conflicts may arise



Git – 3-way Merge

- General process

1. Merge Main into Some Feature branch

```
$ git checkout some-feature  
$ git merge main
```

2. Test branch, fix errors, commit branch

```
(run, edit, modify)  
$ git add changed1.c changed2.c  
$ git commit
```

Do steps 1,2 first

Allows you to debug branch with latest version of Main safely, without breaking Main.

3. Merge Some Feature branch into Main

```
$ git checkout main  
$ git merge some-feature
```

Do steps 3,4 second

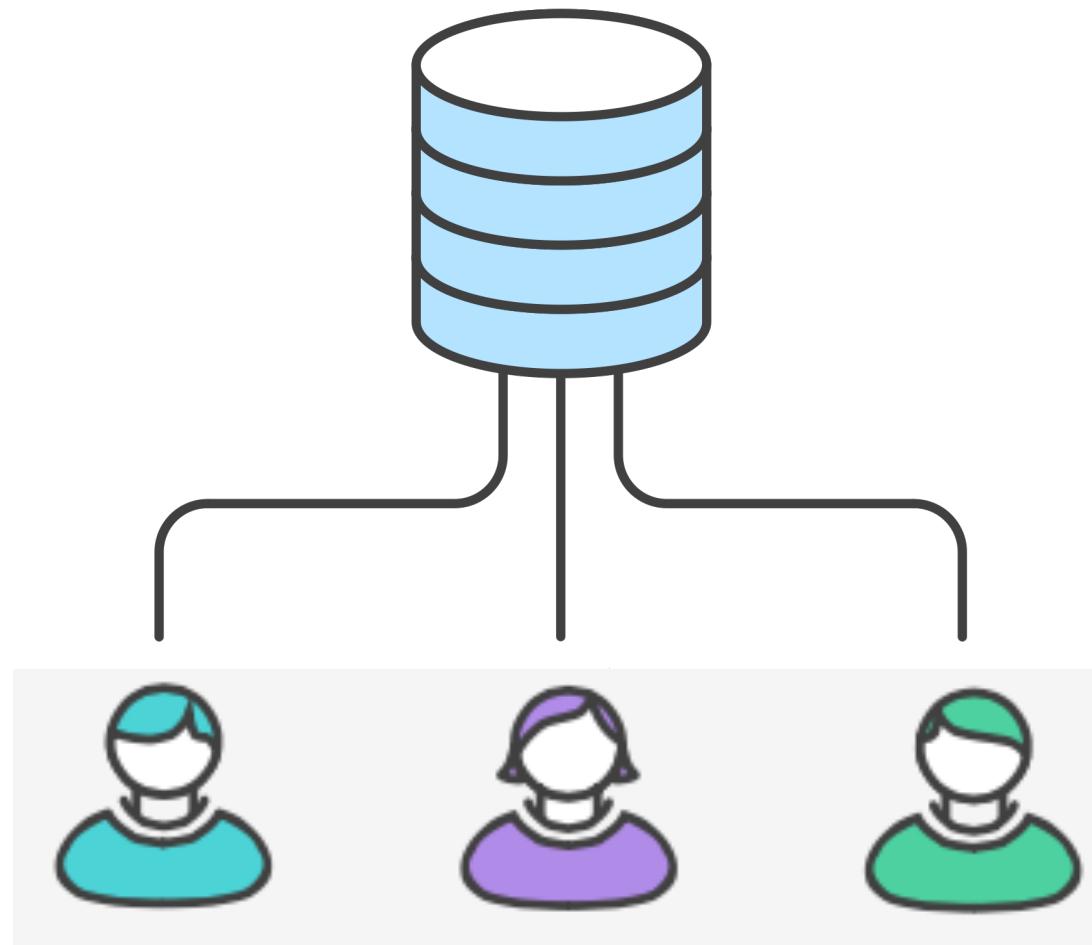
4. Test Main, fix errors, commit Main

```
(run, edit, modify)  
$ git add changed3.c changed4.c  
$ git commit
```

Since branch was just tested, and there were no further changes to Main, there should be no problems here.

Git – Workflow

Everybody clones the central repository



Working in parallel...

John works on his feature



Mary works on her feature

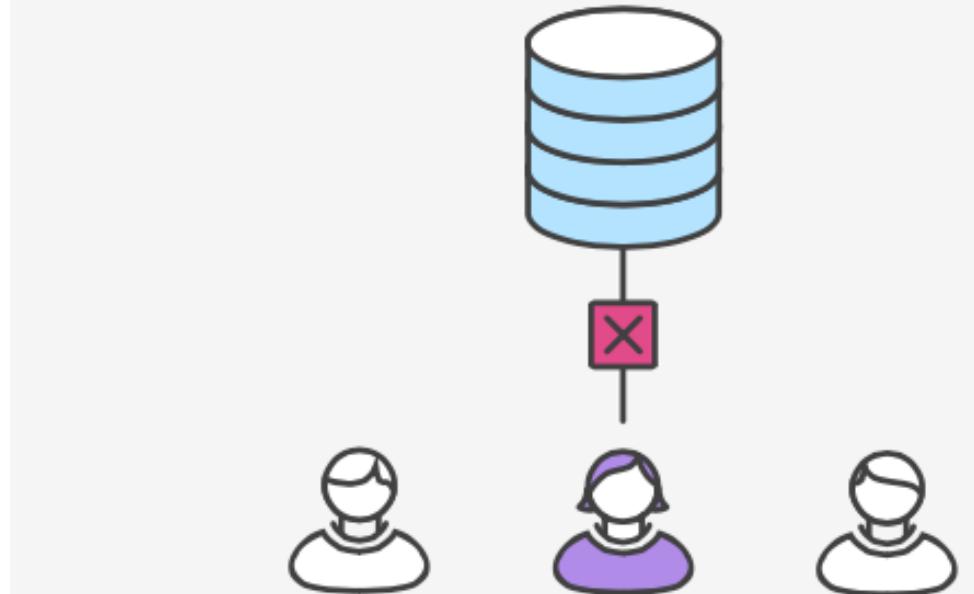


```
(john) $ git push origin main  
(mary) $ git push origin main
```

John publishes his feature



Mary tries to publish her feature

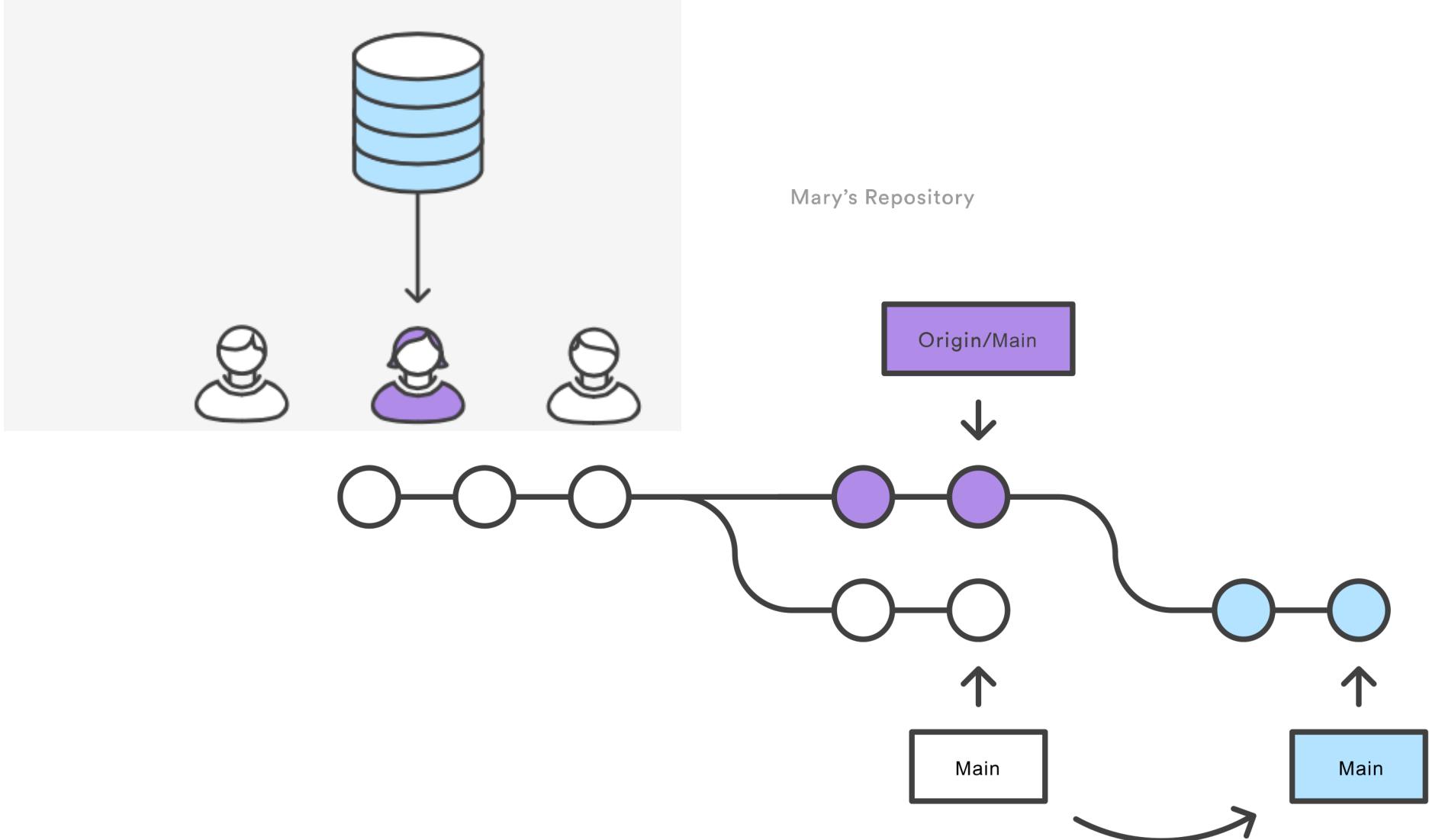


```
error: failed to push some refs to '/path/to/repo.git'  
hint: Updates were rejected because the tip of your current branch is behind  
hint: its remote counterpart. Merge the remote changes (e.g. 'git pull')  
hint: before pushing again.  
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Mary moves her work “after” new Origin/Main

```
(mary) $ git pull --rebase origin main
```

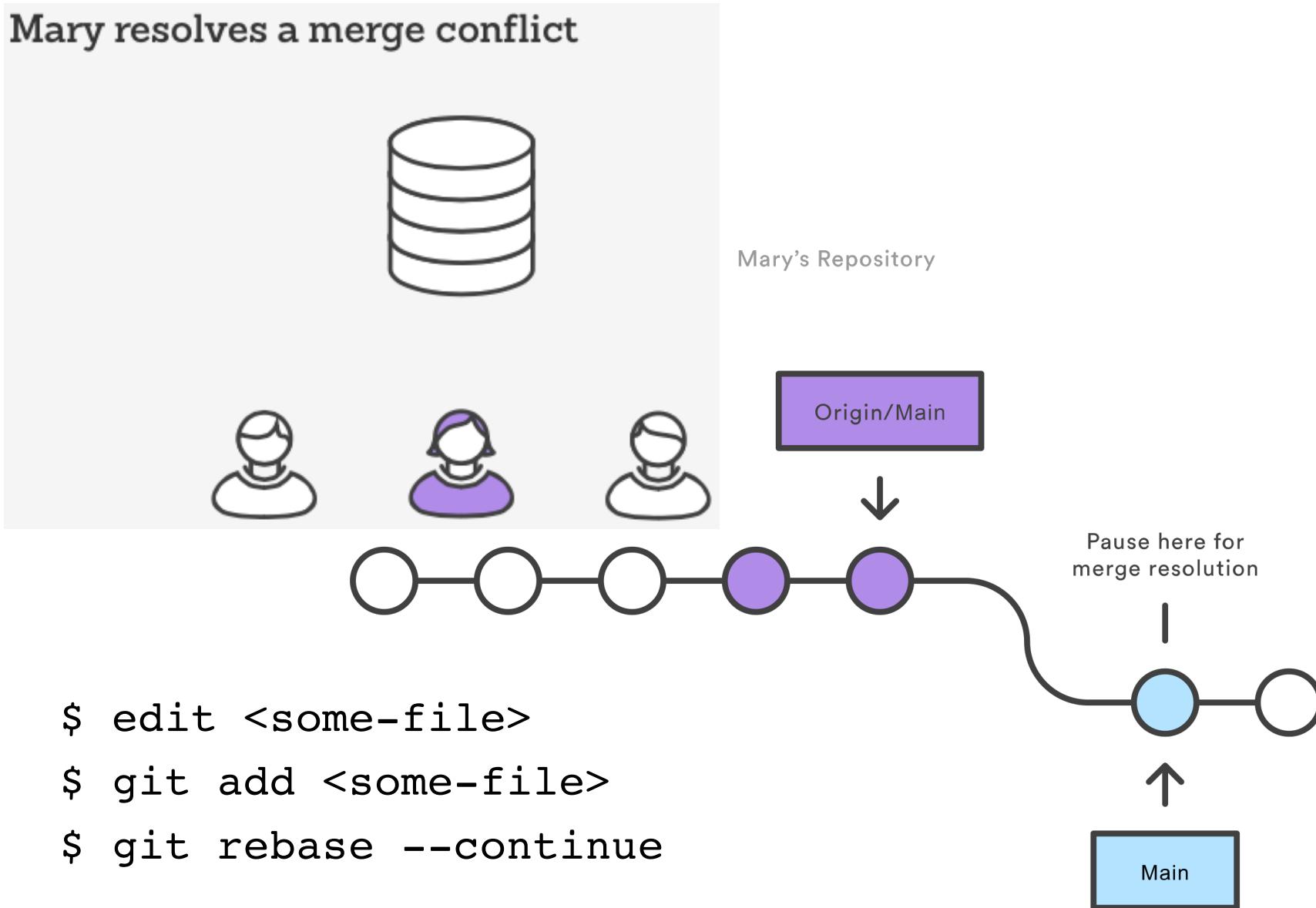
Mary rebases on top of John's commit(s)



Merge Conflicts – Manual Resolution

CONFLICT (content): Merge conflict in <some-file>

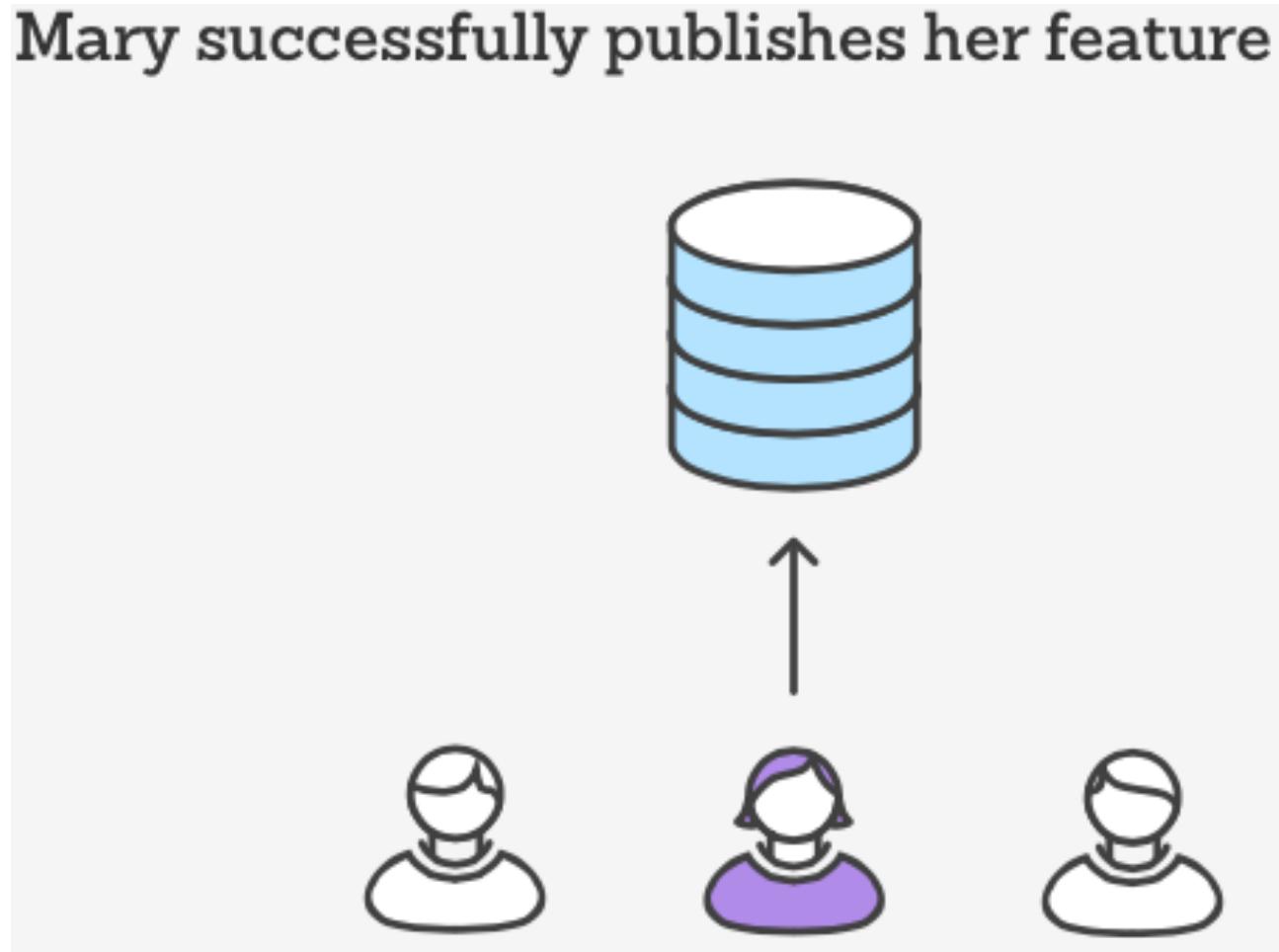
Mary resolves a merge conflict



After Mary Has Successfully Integrated John's Push

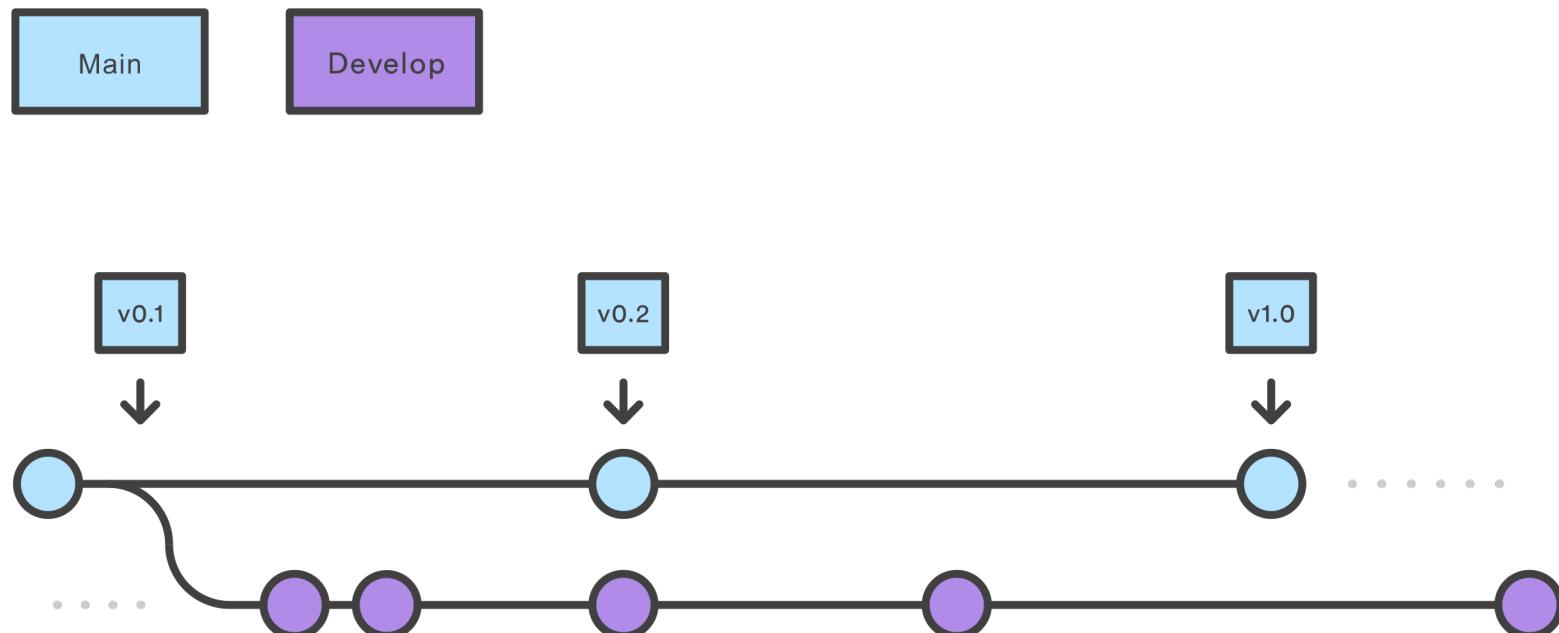
```
(mary) $ git push origin main
```

Mary successfully publishes her feature



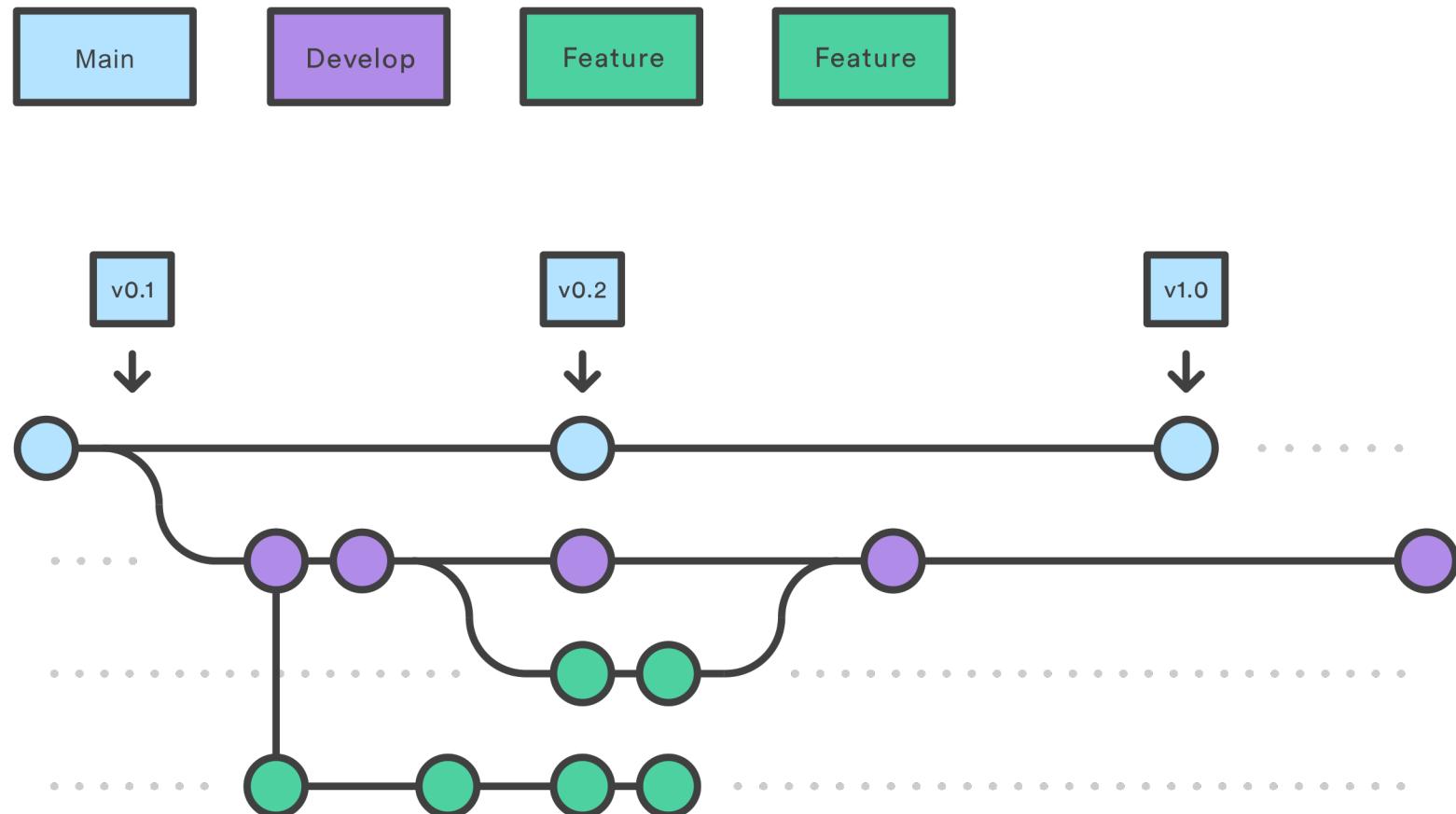
Git –Workflow

- Never work directly in Main branch
 - This is only used to hold released versions
 - Any version in Main ALWAYS works (official release, no bugs)
- Immediately create a “develop” branch
 - Create feature branches off “develop”
 - Merge completed features back in to “develop”



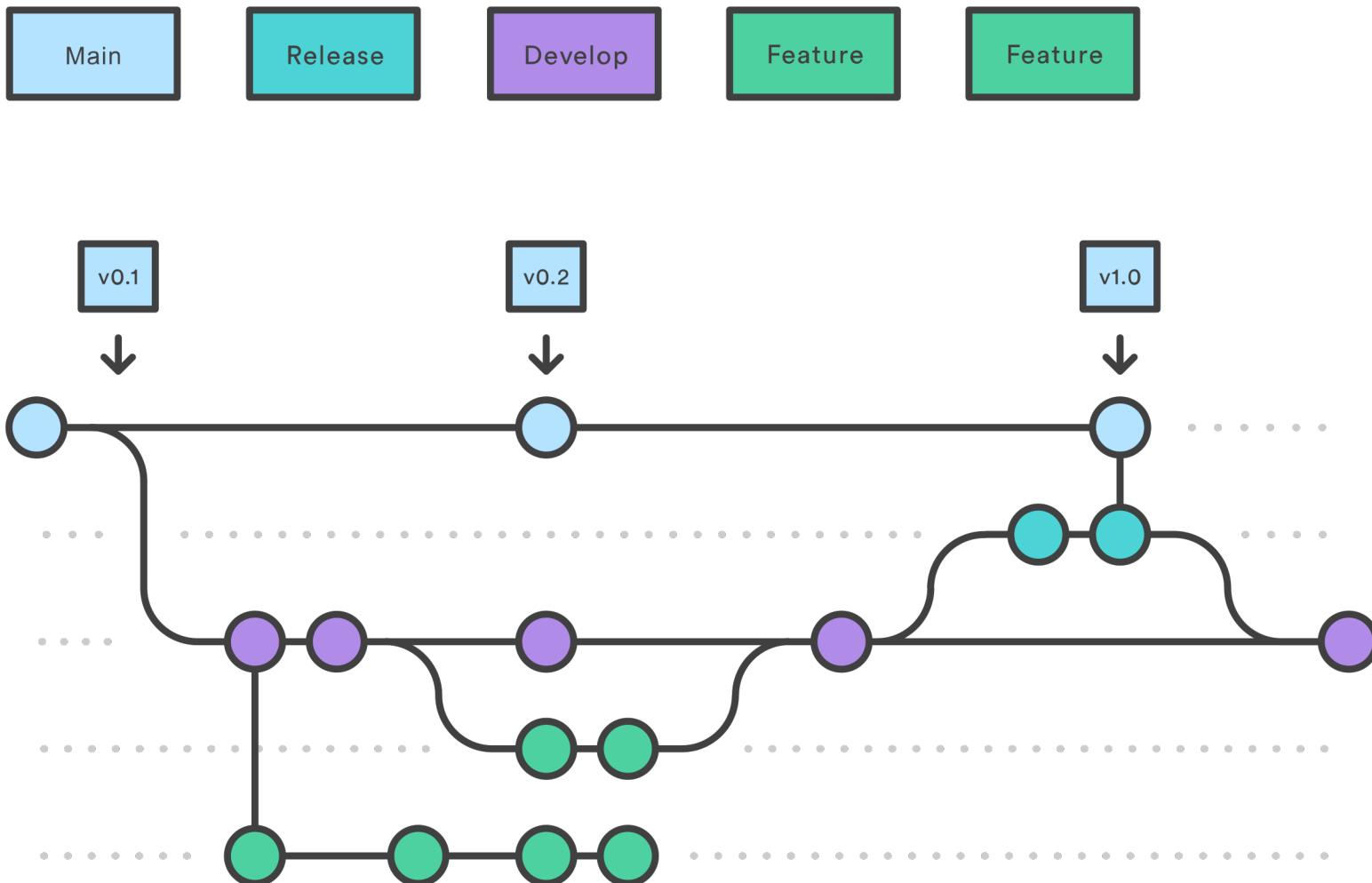
Features as Branches

- Feature branches always fork off of “develop”
 - And merge them back into “develop” as well



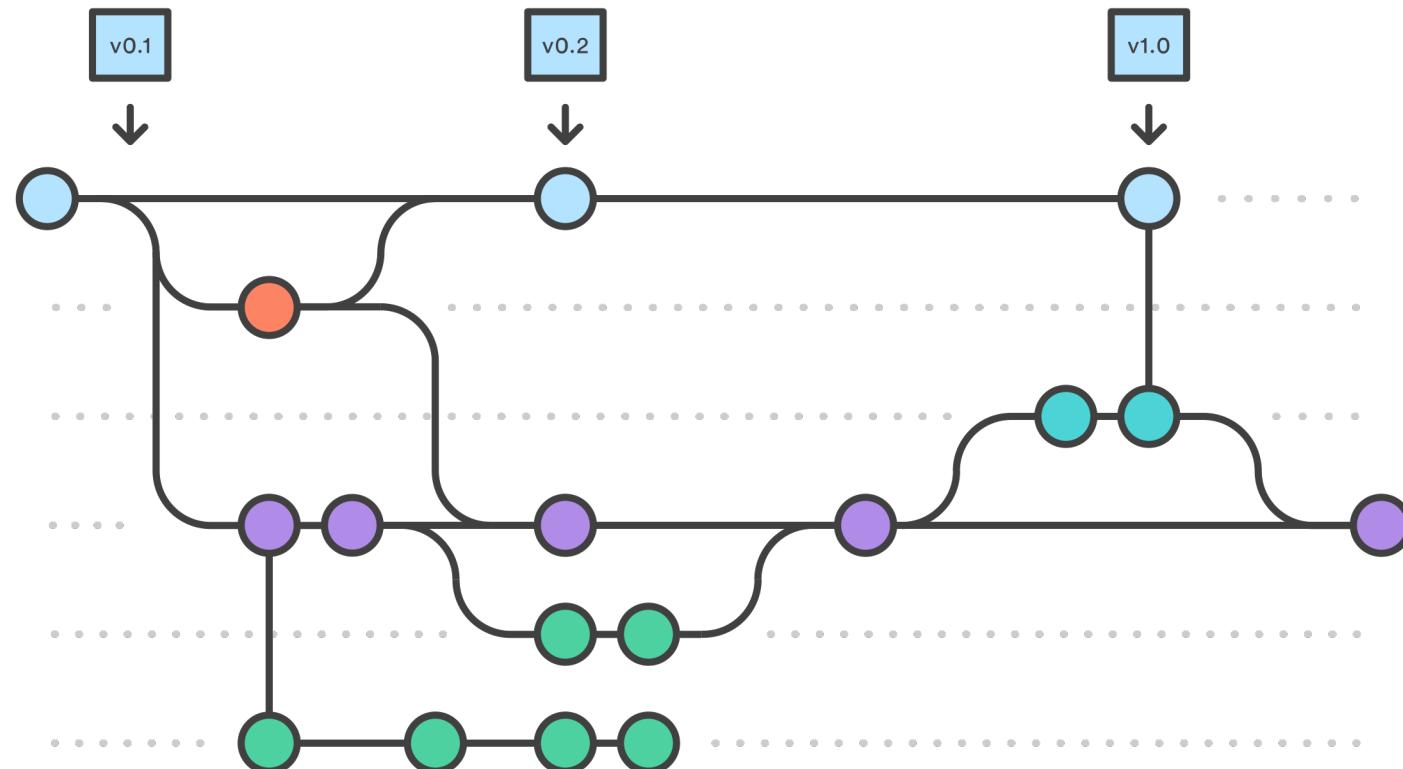
Release Candidates as a Branch

- Create “Release Candidate” branches
 - Merge these into Main when ready



Quick Bugfixes as a Branch

- Maintenance branches, or “hotfixes”
 - Branch off Main
 - Merge into both Main and “develop”



Git – Best Practices

1. Update/pull every morning

- Keeps your local copy inline with revisions done by your team
- Forces you to test your version against the latest version
- For some bugs, must not pull until it's fixed!

2. Commit/push early and often

- Do not code for 2 weeks then expect to push
- Others will have heavily modified repository, your code will be out of date and require many changes

3. Commit logical units

- Each commit should hold the minimal set of files / changes to fix one issue / add one feature
- Use separate commits for different issues/features
- Do not commit individual files, only complete filesets

4. Don't break the build (unit tests must run)

- Always run tests before committing!!!

Git – Hardware and Software

Possible directory structure, all of it in Git!

Note: do not store tool-generated output files in repository

- project
 - docs
 - hardware
 - de1system
 - verilog
 - component1
 - component2
 - testbenches
 - scripts
 - software
 - lib
 - test
 - hwtest
 - src
 - software1
 - software2

Git – More info ?

- Great GIT tutorials
 - <http://www.atlassian.com/git/tutorials>
 - Eg: “Using Branches” and “Comparing Workflows”
- Official documentation
 - <http://git-scm.com/doc>

Pull Requests

- Pushing your changes to origin/main or origin/develop can be destructive
 - Forces others to adopt your changes
 - Your change may break things, prevents others from pushing
- Alternative: **pull request**
 - Email your team, ask them to pull your changes when they are ready
 - This is not actually a git command
 - Also used to ask for feedback
 - Example steps:
 - Push your feature-branch: `git push -u some-feature`
 - Email team with git hash + branch name (some-feature)
 - Feedback requested: member(s) comment on it
 - Merge requested:
 - One member merges/tests/commits into the common shared branch (such as origin/develop)
 - Everyone pulls the committed change

Newer git commands

New git commands to investigate:

- `git worktree`
 - Manage multiple working trees
 - Checks out more than one branch
- `git restore`
 - Restores working tree files
- `git switch`
 - Switches branches