

Intro to Git



Learning Goals

- What is **version control** and **Git**?
- Create your first **repository**
- What is **GitHub**?
- What are **branches** and **pull requests**?
- Make your first contribution to open source

bold = you will know these terms and be able to explain them by the end of this workshop

Leave this room feeling confident that you will be able to use Git and collaborate on a group project

What is **version control**?

Version Control

- A system that records and tracks changes to a set of files
- Allows you to efficiently revert files to an earlier state
- Compare changes over time
- What broke the app?

Think of it as a backup. Roll back to a previous version if you screw up.

Essentially a time machine for your project.

Version Control

- Some analogies – not as robust
 - Google Drive (multiple versions)
 - Adobe Photoshop's History
 - Microsoft Words' 'Track Changes'
- 1972 – SCCS with Unix
- 1982 – RCS
- 1986 – CVS
- 2000 – SVN
- April 2005 - Git

What is **Git**?



Git

- Created by Linus Torvalds <https://www.youtube.com/watch?v=4XpnKHJAok8&t=756s>
- Distributed Version Control
 - No *master* repository*
 - Each user maintains their own code base
 - Encourages participation allowing everyone to work independently
- Essentially stores snapshots of your repository at different points in time

*No such master repository is built into the Git architecture itself.

Let's install (G)it

Install Git - Linux

- Most Linux distributions come natively with Git

```
$ sudo apt-get install git
```

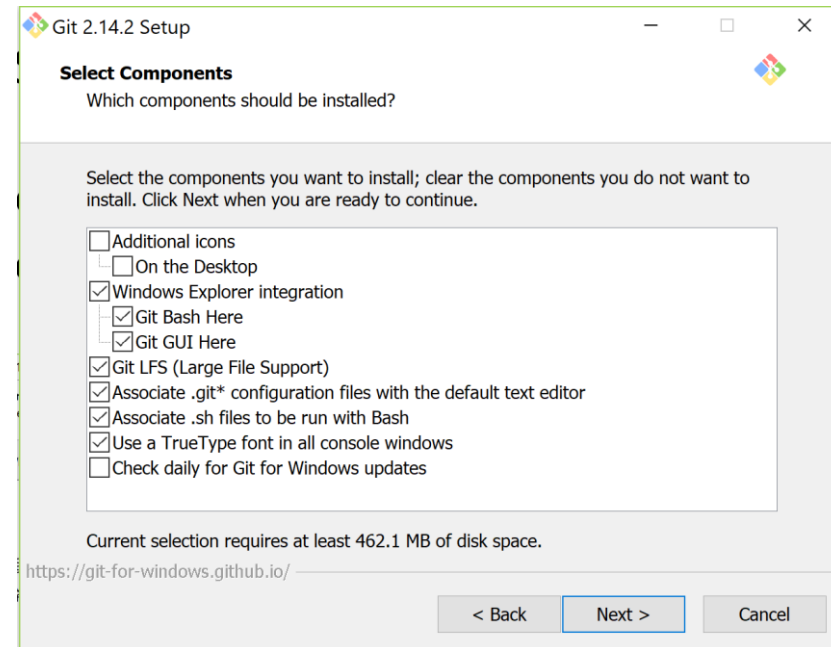
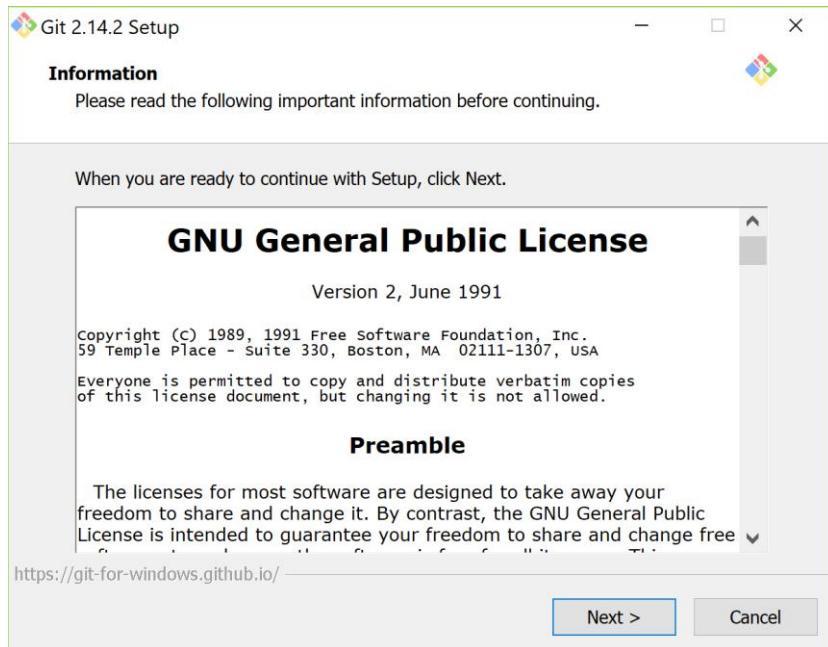
Install Git - MacOS

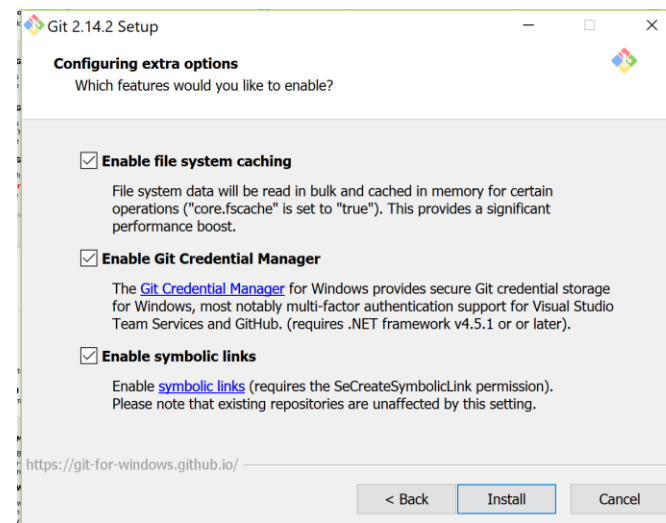
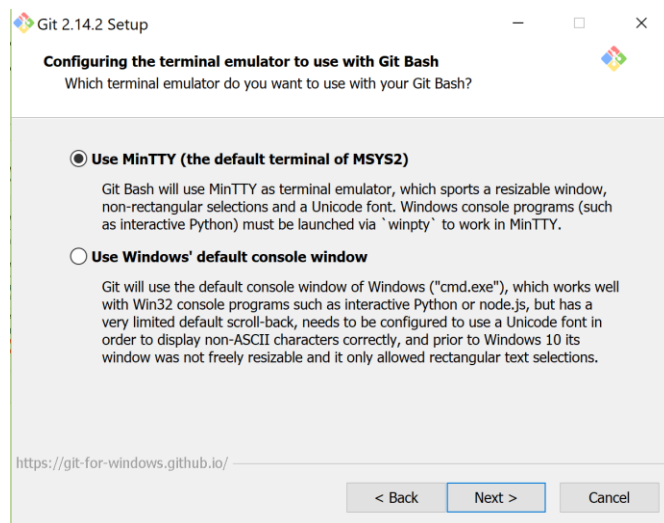
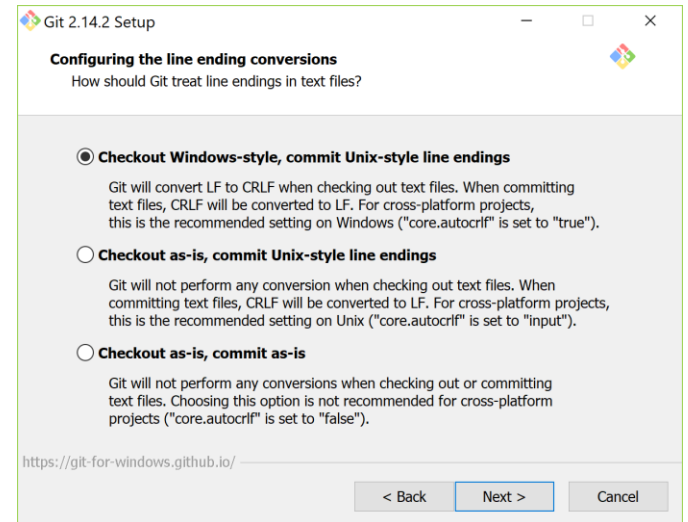
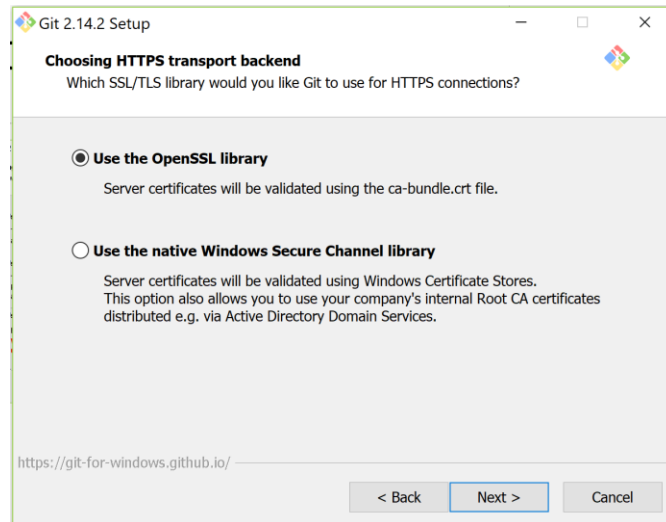
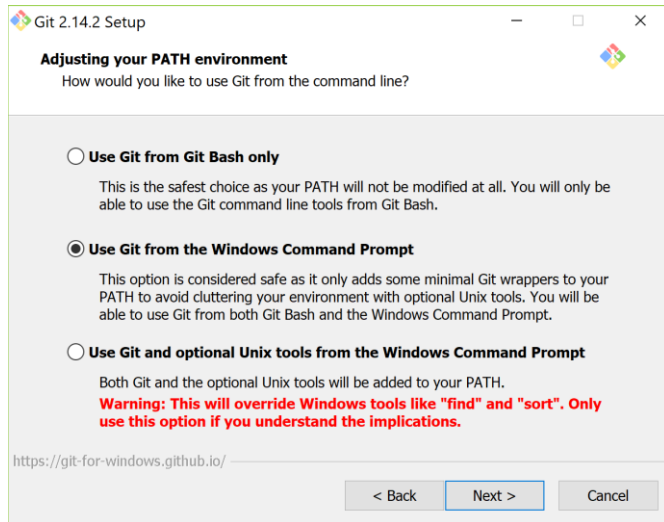
- Install Homebrew from <https://brew.sh/>
- Install Git with brew

```
$ brew install git
```

Install Git - Windows

- Go to <https://git-scm.com/download/win>
- Download **64-bit Git for Windows Setup**





git config

```
$ git config --global user.name "Apara V"
```

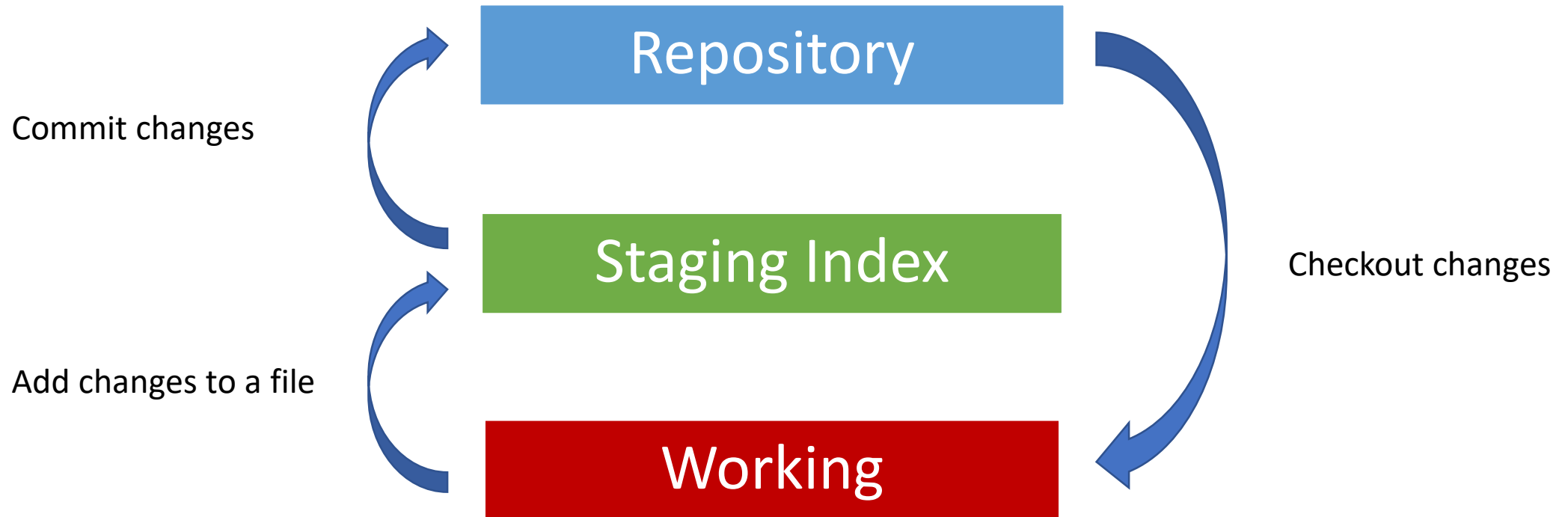
```
$ git config --global user.email "name@domain.com"
```

```
$ git config --global core.editor "notepad.exe"
```

- Git(Hub) makes use of your name and email in the *commits*
- Automatically opens up your editor *when it needs to*
- Use `git config` to create shorter versions of long *commands*

But, how does it work?

The Three-Tree Architecture



The basic commands

- Initialize your repository – ask Git to start tracking files

```
$ git init
```

- Check status of your changes – are they staged or not

```
$ git status
```

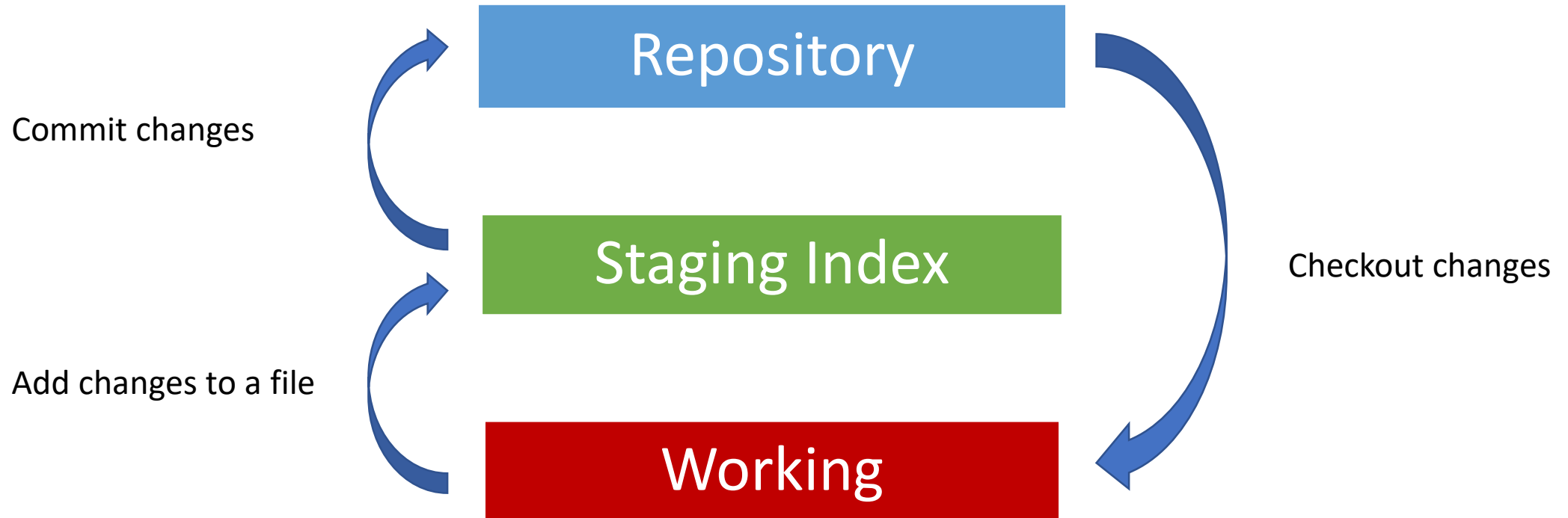
- Add your changes to Staging Index

```
$ git add [files-to-stage]
```

- Commit changes to your repository

```
$ git commit -m “commit message”
```

The Three-Tree Architecture



See it in action

Hello Git!

We need a plan of action!

1. Make a new folder
2. Initialize a repository in this folder
3. Create a file introducing yourself
4. Check status of the repository
5. Commit the file
6. Make some changes to this file
7. Commit these changes

Hello Git!

```
$ mkdir hello_git
```

```
$ cd hello_git
```

```
$ git init .
```

```
$ touch hello.txt
```

Edit this file

Hello Git!

```
$ git status
```

```
$ git add hello.txt
```

```
$ git status
```

```
$ git commit -m "Introduce myself"
```

Note the style of commit message. They should be imperative.

Make changes.

Repeat cycle.



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT
MESSAGES GET LESS AND LESS INFORMATIVE.

Some important commands

- Check your commit history – very important when collaborating with your team

```
$ git log
```

- Checkout a snapshot of your repository

```
$ git checkout <commit>
```

- Compare changes to files

```
$ git diff [file]
```

.gitignore

- Ignore unwanted files

```
$ touch .gitignore
```

```
$ touch unwanted_file.txt
```

```
$ git status
```

Open .gitignore and add unwanted_file.txt to it

```
$ git status
```

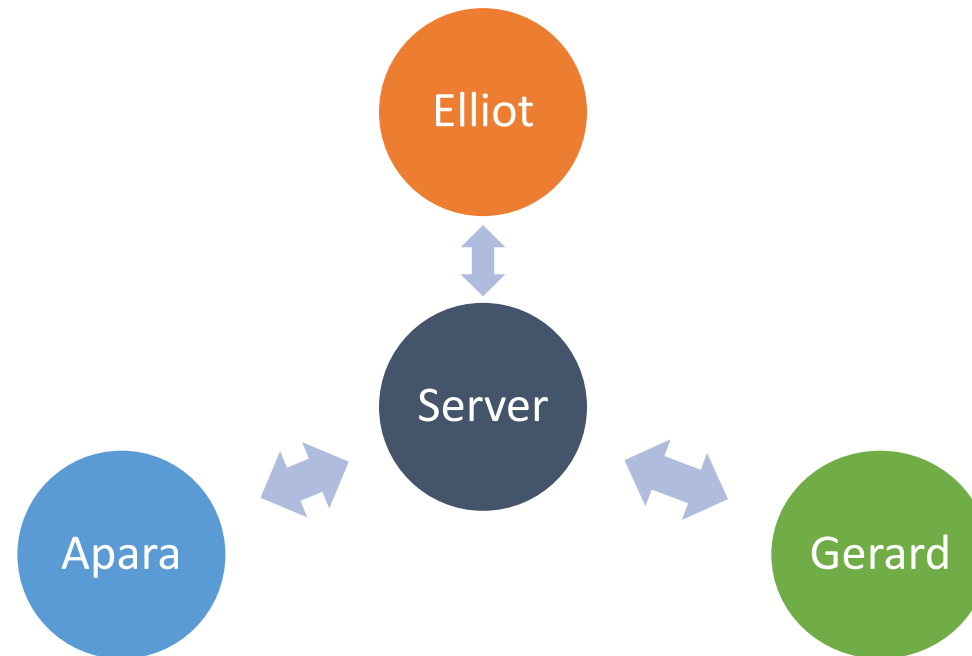
GitHub

GitHub

- A web based Git service
- Offers functionalities of Git and more
- Host all of your code
- In April 2017
 - 20 million users!
 - 57 million repositories!!!
- Basically the party ground for open source communities

Collaboration

- Now we have a remote server where we can store our code
- Opens up opportunities for collaborating with others



Collaboration

- Push your commits to remote server

```
$ git push
```

- Pull other people's commits from remote server

```
$ git pull
```

Push your repository to GitHub

Our plan of action

1. Create a GitHub account
2. Create a new repository
3. Use the commands GitHub suggests to push our repository for everyone to see

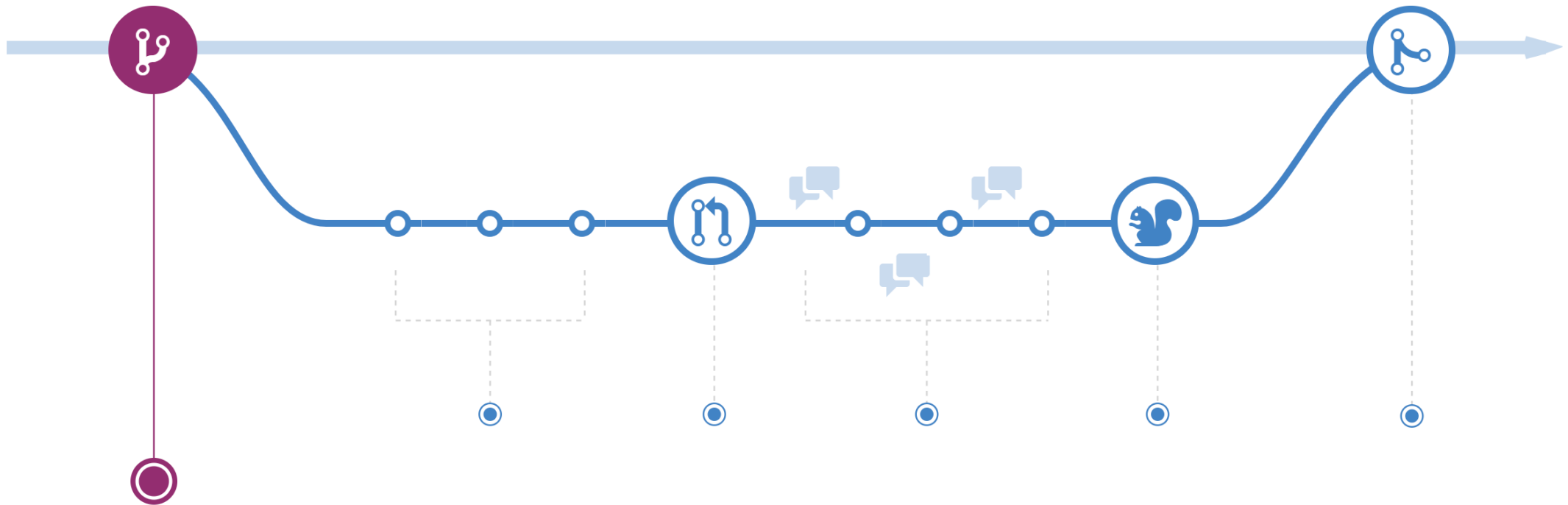
Let's do this together

Branches

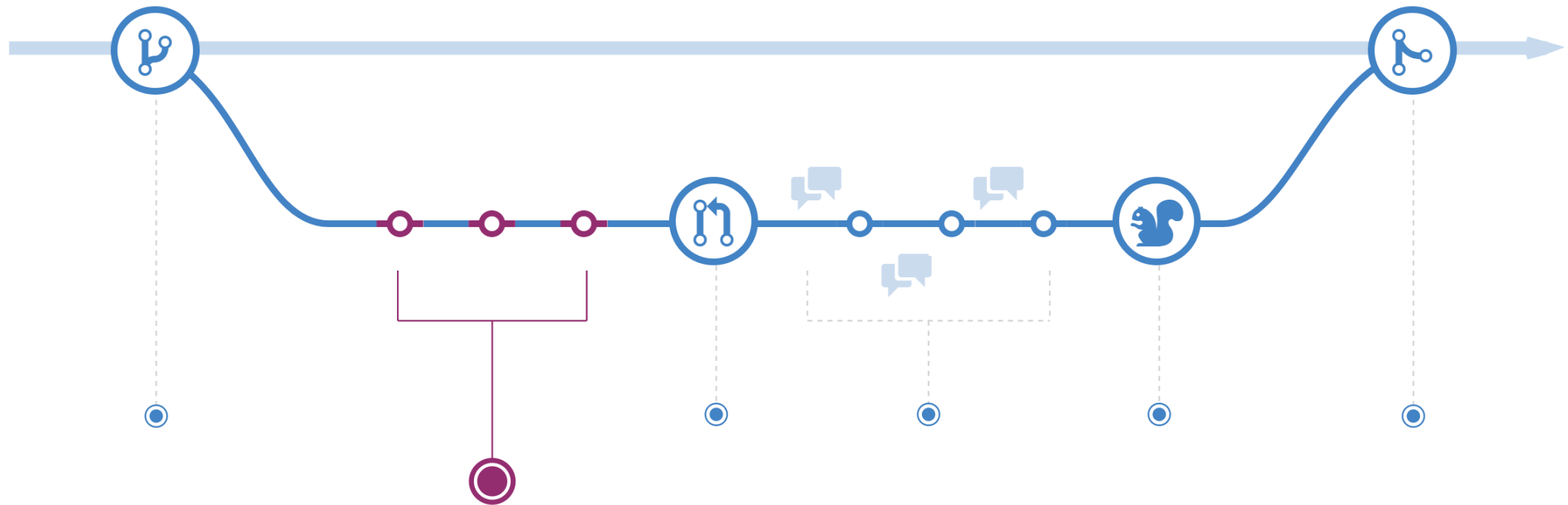
- Diverge from the main line of development and continue to work without changing the main code base
- Essentially branch off the main code base a.k.a. the *master branch*
- Why?
 - Don't want to ruin the production code with your experimental feature
 - Different people writing code at the same time

Branch off -> Work on a set of new features -> Merge these into master
Voila! Release the next update!

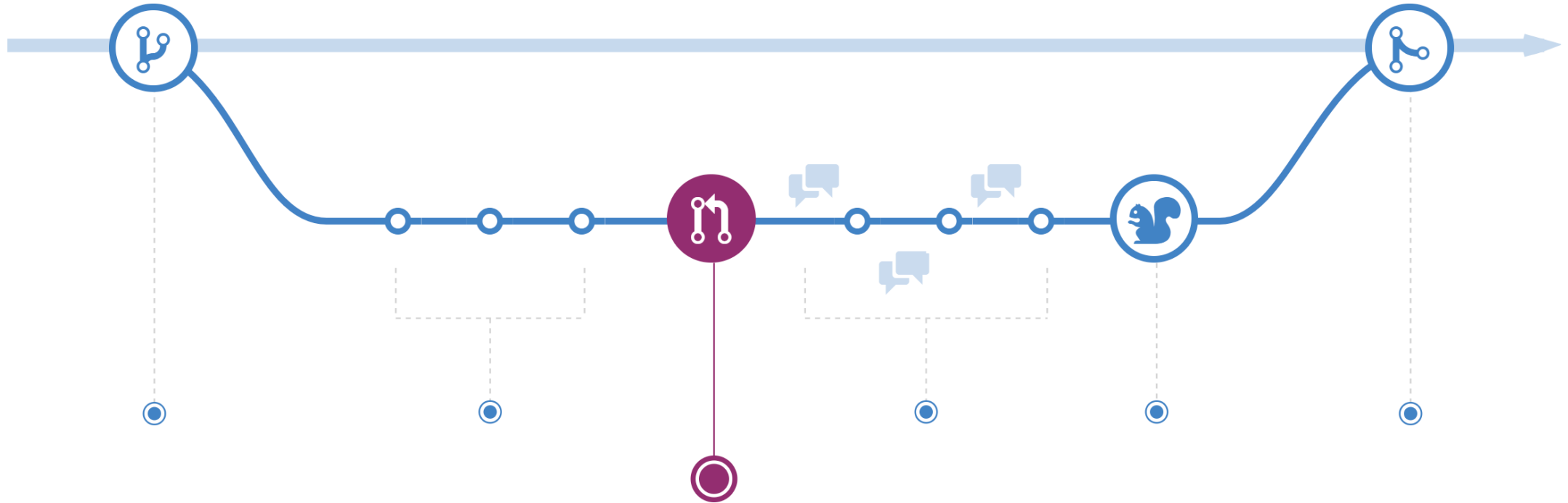
GitHub Flow



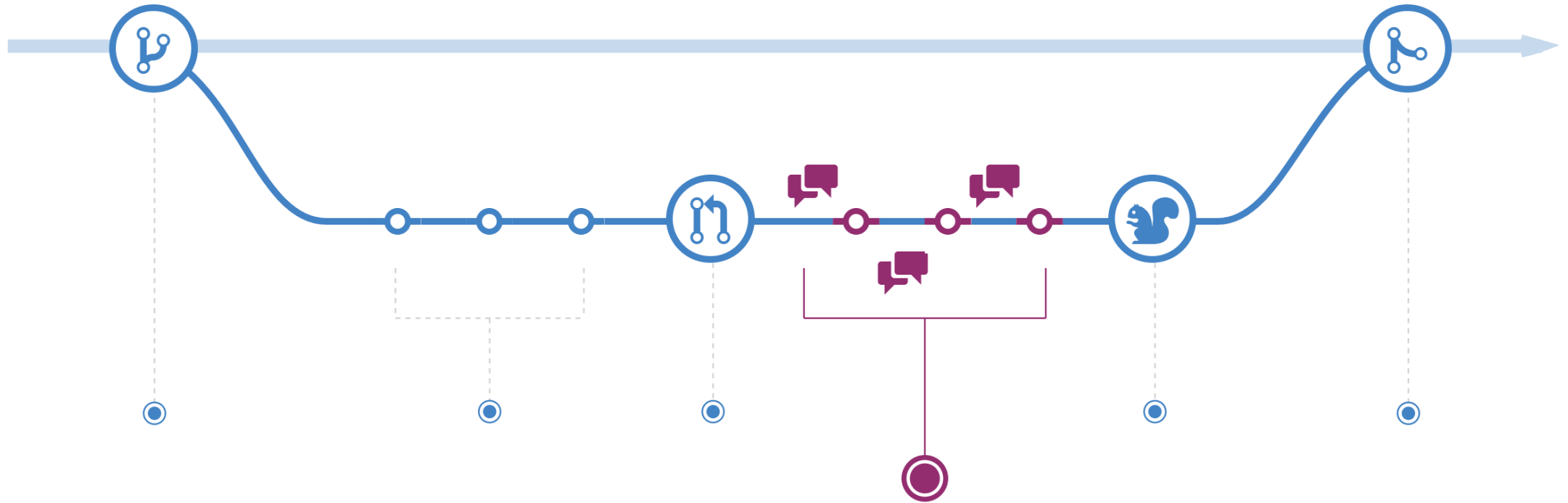
GitHub Flow



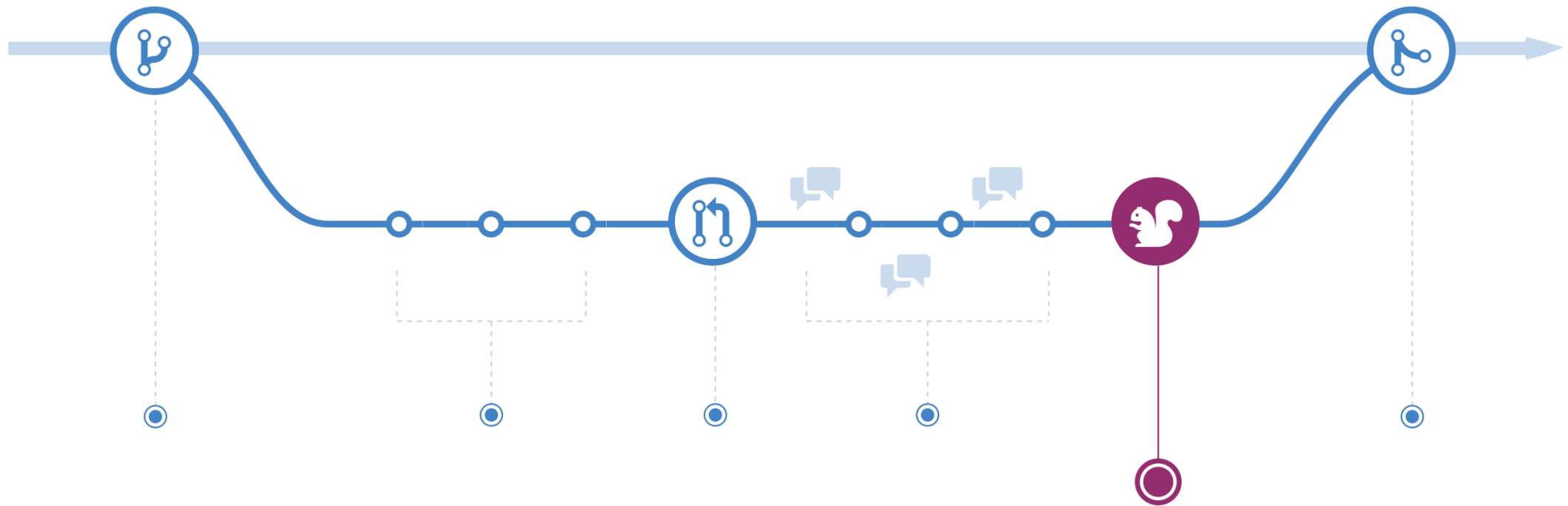
GitHub Flow



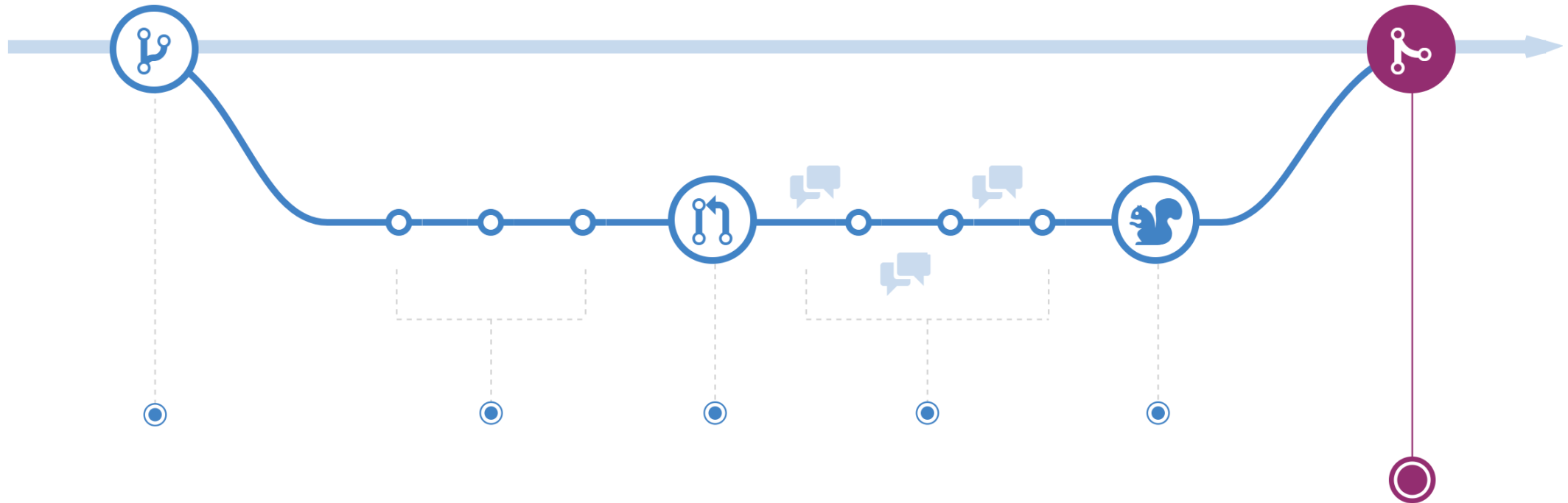
GitHub Flow



GitHub Flow



GitHub Flow



More commands

- Create a new branch

```
$ git branch new_branch
```

- Checkout an existing branch

```
$ git checkout new_branch
```

- Create a new branch and check it out

```
$ git checkout -b newer_branch
```

Merging branches

- Checkout the branch into which you want to merge
- Then `$ git merge branch_to_merge`

Example: I want to merge dev into master

```
$ git checkout master
```

```
$ git merge dev
```

Pull Requests

- Merge Request = Pull Request
- A GitHub feature
- Make process of merging branches a breeze
- You request your changes to be merged to another branch
- Goes through a process of review and changes are evaluated
- Code owners request changes
- Eventually your Pull Request gets merged (or closed 😞)

* Pull Request is colloquially referred to as PR

Take a moment to consider
what you have learnt

- Version control
- Git architecture
- Git commands
- Git workflow
- How to open source your code

That's a lot! Congratulations! Now let's use Git to collaborate!

Let's open a PR

Make your first contribution

- Fork the repo
 - Essentially your account has it's own version of the original repo
- Clone the fork locally
 - Make your version of the repo available to you on your computer

```
$ git clone fork_url
```

Make your first contribution

- Checkout a new branch

```
$ git checkout -b your_name-changes
```

- Add your files

- Commit and push to your fork

```
$ git add .
```

```
$ git commit -m <msg>
```

```
$ git push origin your_name-changes
```

*Note the variant of `git push` command used here

Make your first contribution

- Create your pull request and we will merge it ASAP

What's next?

- Use your Git skills to work on a project with your friend

Get help!

- Join the HackCU community
 - <https://slack.hackcu.org>
 - <https://community.hackcu.org>
- Get the *Pro Git* at <https://git-scm.com/book/en/v2>
- Get your handy Git cheat sheet
 - <https://education.github.com/git-cheat-sheet-education.pdf>

Advanced Git Workshop

There is an Advanced Git workshop in the works

Dive deeper into Git and get dirty

Other workshops

Other workshops are also in the works

Launch your personal website

Learn technologies like Docker, Amazon Alexa, and more...

Hackathons!

Use your Git skills at hackathons

Local Hack Day is coming up on Dec 2nd at Idea Forge

HackCU IV is coming up in February at CU Boulder

Sign up at <https://hackcu.org> to be notified

Thanks to GitHub