

Version Control with git

Session 1: Intro and Basics of git

Nathan Barron

✉ nathanbarron@ou.edu

Why version control?

Getting Started: git and Github

git is an open-source distributed version control system that tracks changes in any set of computer files.

Github is an online host for source code. Github provides tools for using git, collaborating with others, and making code publically available.

git

[Download git for Windows](#)

[Download git for Mac](#)

Github

[Create GitHub Account](#)

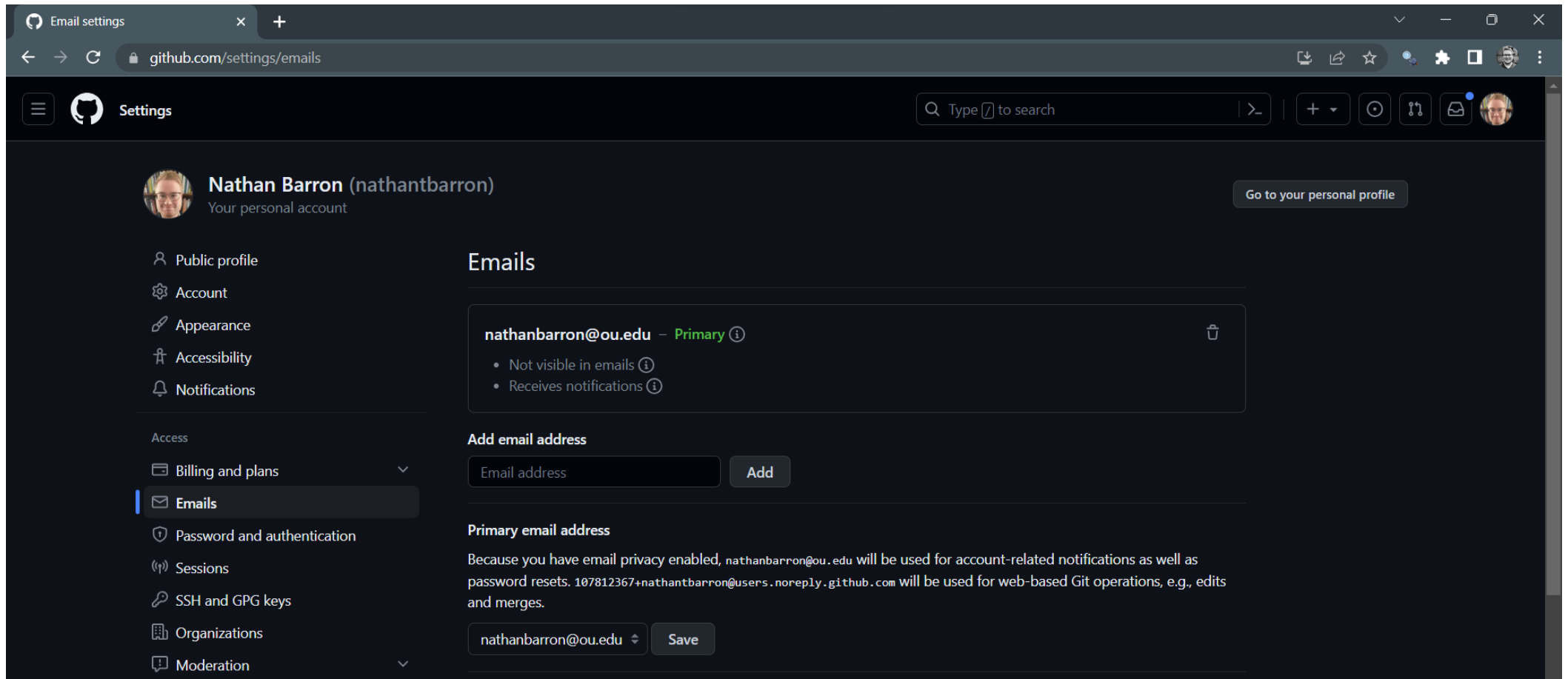
After downloading, open your terminal and run the following code:

```
$ git -v  
$ git
```

Getting Started: Configure git

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

You can anonymize your contributions by setting the `user.email` through a remote email available through GitHub.



The screenshot shows the GitHub 'Email settings' page for user Nathan Barron. The page is dark-themed. On the left, a sidebar contains navigation links: Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails (highlighted), Password and authentication, Sessions, SSH and GPG keys, Organizations, and Moderation. The main content area is titled 'Emails' and shows a list of email addresses. One email, 'nathanbarron@ou.edu', is listed as 'Primary' and has a trash icon. Below this, there is a section 'Add email address' with a text input field and an 'Add' button. Another section, 'Primary email address', contains a text input field with 'nathanbarron@ou.edu' and a 'Save' button. A note explains that the primary email is used for notifications and web-based Git operations.

Email settings

github.com/settings/emails

Settings

Nathan Barron (nathantbarron)
Your personal account

Go to your personal profile

Public profile
Account
Appearance
Accessibility
Notifications

Access

Billing and plans
Emails
Password and authentication
Sessions
SSH and GPG keys
Organizations
Moderation

Emails

nathanbarron@ou.edu - Primary ⓘ

- Not visible in emails ⓘ
- Receives notifications ⓘ

Add email address

Email address Add

Primary email address

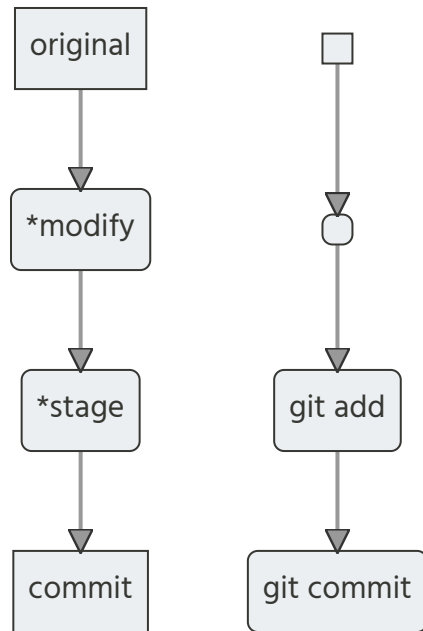
Because you have email privacy enabled, nathanbarron@ou.edu will be used for account-related notifications as well as password resets. 107812367+nathantbarron@users.noreply.github.com will be used for web-based Git operations, e.g., edits and merges.

nathanbarron@ou.edu Save

Basic terminology

- **Files** are the individual objects on a computer that store data, information, settings, or commands used with a computer program.
- A **filesystem** is the way in which files are named and where they are placed logically for storage and retrieval.
- Filesystems often contain **folders**, which are containers used to contain logically-related files within a filesystem.
- The **working directory / workspace** is the top-level project folder that contains all content available in the filesystem.
- A **repository** is the `.git/` folder inside your working directory. The repository (or 'repo') tracks changes.

Basic workflow in local repository



1. **Begin with an original workspace.**

The original state may be an empty workspace or a fully-developed workspace with many existing files.

2. **Make a modification in the workspace.**

The modification can be the creation of a new file, deletion of an existing file, or some change of an existing file.

!!!*git tracks files within a workspace.*

3. **Stage the modification(s).**

Once changes have been made, you'll want to select specific changes to group together. (Ideally, they would be logically related to each other.)

4. **Commit the staged modifications.**

Take the batch of staged modifications and commit those changes to the core version of your current branch (e.g. 'main').

Create a workspace and repository

Remember: A workspace is *not* the same thing as a repository.

```
$ cd
$ cd desktop
$ mkdir myrepo
$ cd myrepo
$ git init
$ git status

$ type nul > README.md # Windows
$ type nul > README.md # Windows

$ touch README.md # Mac
$ touch README.md # Mac

$ git status

$ git add .
$ git status

$ git commit -m "First commit"
$ git status
```

Tracking changes and checking history

Track changes

```
$ git restore <file>
$ git restore --staged <file>

$ git diff # change in repo (not staged)
$ git diff --staged # change only in staged
```

Explore history

```
$ git log
$ git log --oneline
```

Reverting to previous versions

```
git diff HEAD <file>
git diff <file>
git diff HEAD~1 <file>
git diff HEAD~2 <file>

git show HEAD~2 <file>

git log
git show <long-id> <file>
git log --oneline
git show <short-id> <file>

git checkout <short-id> <file>
git checkout HEAD <file>

git checkout <short-id>
git checkout main
```


Ignore things

There may be some files that you don't want to track called *artifacts* (or, files automatically generated during a development/rendering process). You might also not want to track large datasets that are being continuously updated. You can specify which files or types of files you want to ignore in a **.gitignore** file. When starting a Github repo, you are given the option to include template .gitignore files.

.gitignore template for R

```
# History files
.Rhistory
.Rapp.history

# Session Data files
.RData
.RDataTmp

# User-specific files
.Ruserdata

# Example code in package build process
*-Ex.R

# Output files from R CMD build
/*tar.gz

# Output files from R CMD check
/*Rcheck/

# RStudio files
.Rproj.user/

# produced vignettes
vignettes/*html
```

.gitignore template for Python

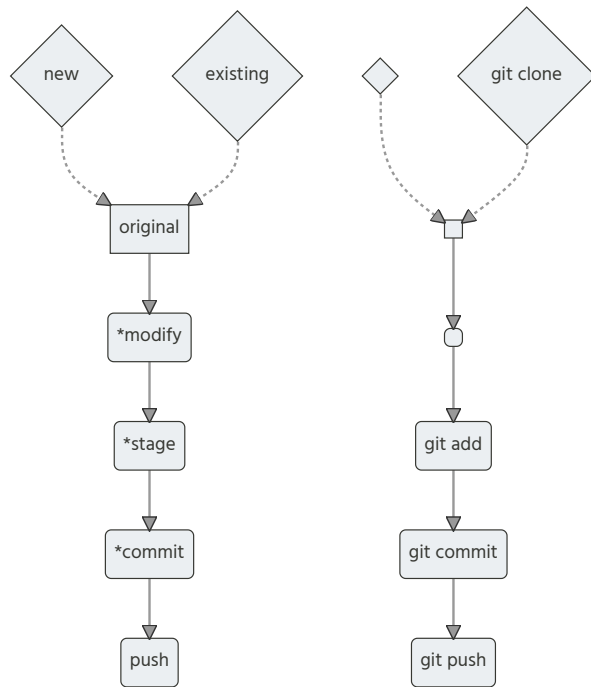
```
# Byte-compiled / optimized / DLL files
__pycache__ /
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
installled.cfg
```

Pushing to Github

Updated basic workflow with remote repository



The **original** state is defined by file contents at the last push (i.e. the most recent branch version on Github).

The **modification** can be the creation, deletion, or modification of an existing file.

The **staging** (git add) separates files you would like to include in a single commit from those you do not want to include in the commit.

The **commit** (git commit) officially logs the changes in the *local* repository and makes those changes eligible to get pushed to the remote repository.

The **push** (git push) sends a copy of the local repository (and the corresponding .git history files) to the remote repository (Github)

Creating a New Github Repo

1. Create an empty Github repo
2. Grab [https:// link](https://link)
3. Initialize local repository

```
cd <workspace_path> # Set cd to the directory path
echo "#existing" >> README.md # Create README.md file
git init
git add . # The '.' means 'all eligible items'
git commit -m "first commit"
git branch -M main # This will ensure you're on the main branch
```

3. Modify, Stage, Commit

4. Push to Github

```
git remote add origin <https://link>
git push -u origin main
```

After the initial push, you should be able to push with only `git push`, but you can make sure you're pushing to the right location with `git remote -v`

Cloning an Existing Github Repo

1. Grab [https://](https://github.com/nathantbarron/git-demo.git) link
2. Clone the repository

```
cd Desktop # navigate to Desktop
git clone https://github.com/nathantbarron/git-demo.git
cd git-demo
git status
git log
```

3. Modify, Stage, Commit

4. Push to Github

```
type nul > newfile.txt
git add newfile.txt
git commit -m "first post-clone commit"
git status

git remote -v
git push
```

NOTE For now, we're only talking about cloning a GitHub repository of which you are an owner.

Creating a README.md file

A `README.md` file briefly explains a workspace, including the purpose of the code, structure of the filesystem, important features, etc. Especially if you're publishing open-source code, you'll want to mention the license attached to your code ([more on licenses here](#)).

Concise, thoughtful, and well-written README.md files are essential to providing open-source code, collaborating with others, or just remembering what you were doing.

The `.md` file extension denotes a *markdown* file, which is a simplified version of HTML that can also render raw HTML commands.

Version Control with git

Session 2: IDE Integration and Collaboration

Nathan Barron

✉ nathanbarron@ou.edu

IDE Integration (Code-along)

Integration with RStudio

Setting up a git project in RStudio

Setting up and using version control in RStudio

Using GitHub as remote storage for your project

Making further changes to your project

What we've learned and further steps

Integration with VS Code

Overview of Source Control in VS Code

Intro to using git in VS Code

Working with GitHub in VS Code

FAQ

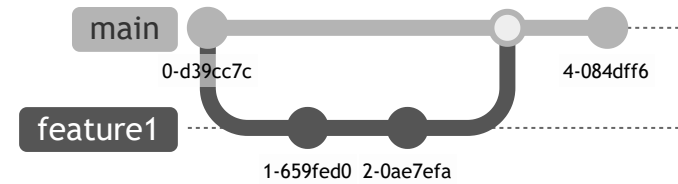
Collaboration with GitHub

Possible Workflows

Suggested Personal Workflow

Don't commit directly to `main`

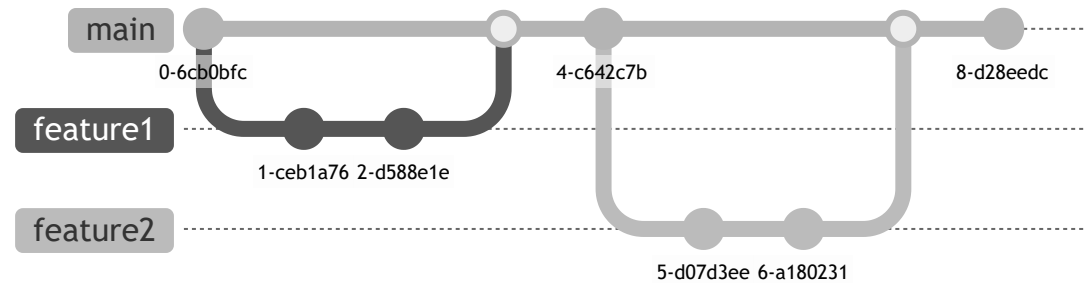
Use branches for updates



Problem: Simultaneous Updates

What is the problem in this case?

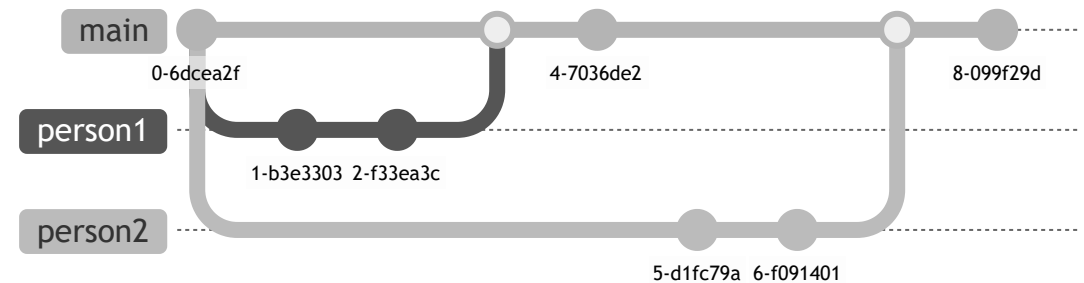
You will often want to work on multiple features at the same time



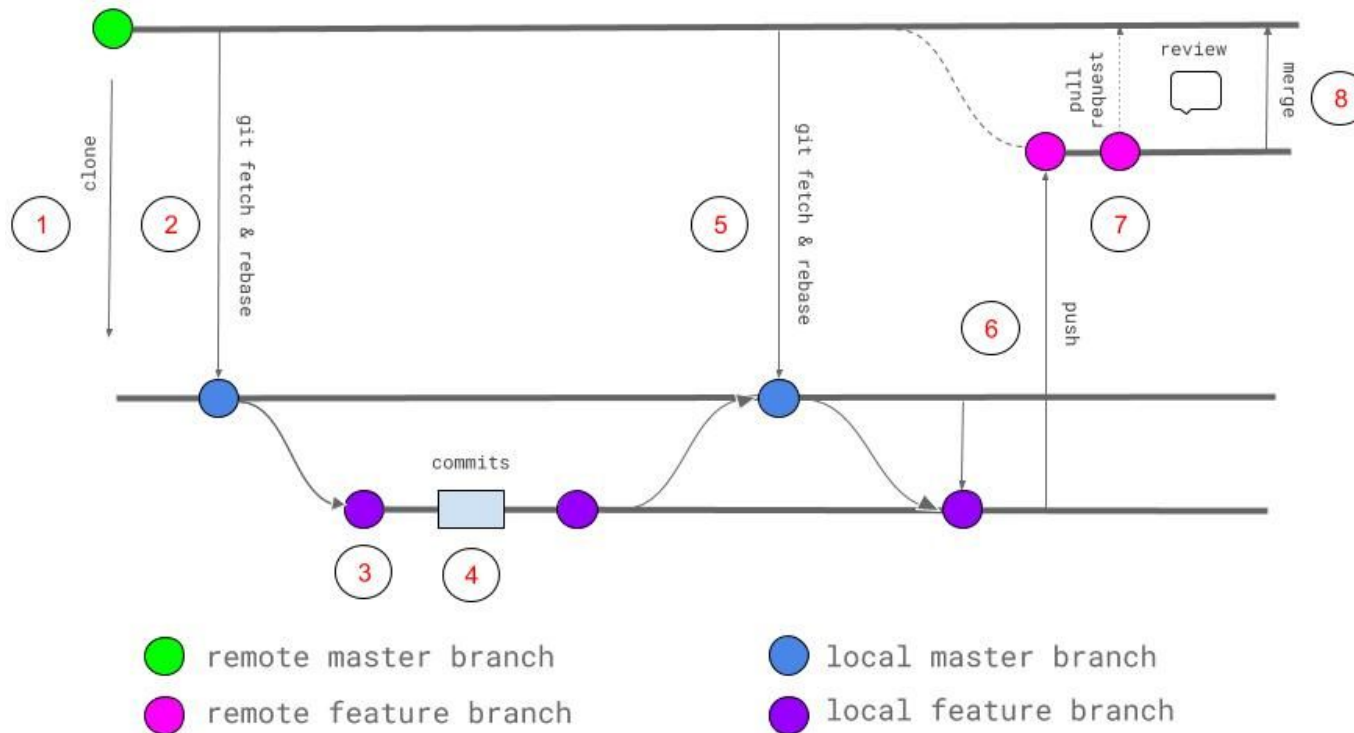
Problem: Collaboration

What is the problem in this case?

person2 is based on an old version and overwrites person1



Safe Collaborative Workflow



Step #8 Discuss, review and merge pull request

1. Clone the repository
2. `git fetch` and `git rebase` (or `git pull`)
3. Create new branch
4. Add commits
5. `git fetch` and `git rebase`
6. Push the commits
7. Create a Pull Request
8. Discuss, review, and merge pull request

Terminology

- A **branch** is a separate version of the main repository. The primary branch is usually called **main** (or **master**). Lower branches can be made for additional features, analyses, users, etc.
- A **clone** is a local version of a remote repository. If you clone a repository to which you *do not* have ownership or editing access, you can turn your clone (local) into a **fork** (remote). *There is no formal **fork** command. Usually repos are forked prior to cloning.*
- The **pull** command fetches and downloads content from a remote repository to update the local repository. (akin to **git fetch** + **git rebase**)
- A **pull request** is made when a user wants to push changes to a branch and either (1) does not have editing permission, and/or (2) wants collaborators to review and discuss the addition prior to merging.
- A **conflict** arises when two branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other.

Managing Conflicts

- The best way to prevent merging conflicts is to **always** `git pull` on higher branch prior to `git merge` with higher branch
- However, conflicts can still occur – especially when there are multiple outstanding pull requests
- You can fix merge conflicts directly in GitHub