Princeton University
COS 333: Advanced Programming Techniques
Git and GitHub Primer

Introduction

Git is a distributed version control system. It was created by Linus Torvalds – the same person who created the Linux operating system – in 2005 for the purpose of managing multi-programmer development of the Linux kernel. Using Git you create repositories containing, typically, source code.

GitHub is an Internet service for hosting Git repositories. It was created in 2007 by Chris Wenstrath, P. J. Hyett, Tom Preston-Werner, and Scott Chacon. It's an Internet service for hosting Git repositories.

You must use Git and GitHub for your project. It would be reasonable to use Git and GitHub for your COS 333 *assignments* too, and I highly recommend that you do so.

If you want to use a version control system other than Git (for example, Mercurial or Subversion), or some hosting service other than GitHub (for example, BitBucket), then please discuss the matter with me before doing so.

This document describes a subset of Git and GitHub that may be sufficient for your work in COS 333. The first sections of this document describe setup steps that you should perform near the beginning of the semester. The remaining sections describe common use cases.

Let's say that you want to use Git and GitHub for Assignment 1...

Setup Step 1: Installing Git

Perform this step one time only, near the beginning of the semester.

Install the Git program on the computer that you will use to do software development. The instructions for installing Git depend upon what kind of computer you will use. If your development computer is:

- A courselab computer, then you can skip this step. Git already is installed on courselab.
- Your own Mac computer (running OS X version 10.9 or above), then issue a `git` command in a terminal window. If Git isn't already installed, then OS X will prompt you to install the Xcode command-line tools. Installing the Xcode command-line tools also installs Git.
- Your own computer running Microsoft Windows, then browse to http://git-scm.com/download/win. The download and install will start automatically.
- Your own computer running Linux, then use your package manager to install Git.

Setup Step 2: Configuring Git

Perform this step one time only, near the beginning of the semester.

Configure Git to indicate your identity and preferences. To do that, issue these commands in a terminal window:

```
$ git config --global user.name "yourname"
$ git config --global user.email youremailaddress
$ git config --global color.ui auto
$ git config –-global core.editor yourpreferrededitor
```

For example, I might issue these commands:

```
     $ git config --global user.name "Robert Dondero"
     $ git config --global user.email rdondero@cs.princeton.edu
     $ git config --global color.ui auto
     $ git config –-global core.editor emacs
```

For `youremailaddress` I recommend that you specify your Princeton email address, but it's fine to specify any of your email addresses.

Git uses `yourpreferrededitor` if you issue a `git commit` command without the `-m` option. The `git commit` command is described later in this document.

In response to those commands Git stores your settings in a file named `.gitconfig` in your home directory.

Setup Step 3: Creating a GitHub Account

Perform this step one time only, near the beginning of the semester.

Create a GitHub free account. Doing so will allow you to create private GitHub repositories with an unlimited number of collaborators. To do that, browse to https://github.com. Click on the *Sign up for GitHub* button, and follow the instructions. Eventually your browser displays a *What do you want to do first?* page.

Setup Step 4: Creating a GitHub Repository

Perform this step one time only, near the beginning of the semester.

Continued from the previous step:

- In the *What do you want to do first* page, click on the *Create a repository* button.
- In the resulting page, for *Repository name* enter `cos333asgt1`, select the *Private* radio button, check the *Add a README file* checkbox, and click on the *Create repository* button.
- In the resulting COS333asgt1 page, click on the *Setting* tab, click on *Manage access*, click on the *Invite a collaborator* button, and complete the dialog to invite your teammate to collaborate.

IMPORTANT: Your GitHub **assignment** repositories **must be private** such that only you and your assignment teammate have access to them.

IMPORTANT: I prefer that your GitHub **project** repository be private such that only you, your project teammates, and the COS 333 instructors have access to it. But it's OK to make your GitHub project repository publicly readable if you have a compelling reason to do so.

(Optional) To create additional repositories… Browse to https://github.com/ and sign into your account. Click on the plus sign at the top right of the page. Click on *New Repository*.

Setup Step 5: Creating a Personal Access Token

Perform this step one time only, near the beginning of the semester.

To use GitHub via its CLI (command-line interface), you must create a GitHub *personal access token* (*PAT*). The instructions for doing so are at this page:

https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token

In summary:

- Browse to your GitHub repository page
- Click the profile icon at the upper right to display a drop-down menu.
- In the drop-down menu click on *Settings*
- In the resulting page, click on *Developer Settings*.
- In the resulting page, click on *Personal access tokens*.
- In the resulting page, click on *Generate new token*.
- In the *Note* area, enter "COS 333". Check the *repo* checkbox. Click on the *Generate Token* button.
- The resulting page displays a PAT. It's important to remember your PAT. Copy it to your computer's clipboard, and maybe store it in a file, at least temporarily.

From this point forward you use the GitHub CLI (that is, the GitHub command-line interface, that is, `git` commands issued in a terminal window) to interact with GitHub.

Setup Step 6: Cloning Your GitHub Repository

Perform this step one time only, near the beginning of the semester.

- Open a terminal window on your computer. Issue `cd` commands to change your working directory to the one where you want your development Git repository to reside. Issue this command:

  ```
  git clone https://github.com/yourgithubusername/cos333asgt1.git
  ```

  For example, I might issue this command:

  ```
      $ git clone https://github.com/rdondero/cos333asgt1.git
  ```

  When prompted for a username, enter your GitHub username. When prompted for a password, enter the PAT that you generated in the previous step. The command generates output similar to this:

  ```
  Cloning into 'cos333asgt1'...
  Username for 'https://github.com': yourgithubusername
  Password for 'https://yourgithubusername@github.com':
  remote: Enumerating objects: 3, done.
  remote: Counting objects: 100% (3/3), done.
  remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
  Unpacking objects: 100% (3/3), done.
  ```

  Your working directory contains a new directory named `cos333asgt1` which contains a `README.md` file and a `.git` directory. The `cos333asgt1` directory is your *development repository*.

(Optional) No doubt you will find it inconvenient to enter your GitHub username and PAT each time you issue a `git` command that interacts with GitHub. To avoid the need to do so, issue this command one time:

```
     git config --global credential.helper store
```

In response to that command, Git updates the `.gitconfig` file in your home directory, and stores your GitHub credentials in a file named `.git-credentials` in your home directory. After issuing that command you may need to enter your GitHub username and personal access token one more time when interacting with GitHub. Thereafter, Git automatically uses your stored GitHub username and PAT when interacting with GitHub.

Use Cases

Having completed the setup steps, you now have your own private *GitHub repository* in the GitHub cloud and your own *development repository* on your development computer. Use those repositories as vehicles for learning about Git. In particular, try implementing the use cases given in this document. Experiment!!!

Use Case 1: Adding Files

Perform this step repeatedly throughout the semester as required.

Add *file1* and *file2* to your development repository and your GitHub repository. To do that, in a terminal window on your development computer issue these commands:

```
# cd to the dev repo directory.
cd cos333asgt1

# Pull the GitHub repo to the dev repo.
git pull
```

Use an editor to create *file1* and *file2*.

```
# Stage file1 and file2.
git add file1 file2

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"
```

Note: If you omit the `-m "Message describing the commit"` option, then Git launches the editor that you specified in Setup Step 2, with the expectation that you will compose a message using that editor.

```
# Push the dev repo to the GitHub repo.
git push origin main

# (Optionally) check status.
git status
```

Use Case 2: Changing Files

Perform this step repeatedly throughout the semester as required.

Change *file1* and *file2* in your development repository and your GitHub repository. To do that, in a terminal window on your development computer issue these commands:

```
# cd to the dev repo directory.
cd cos333asgt1

# Pull the GitHub repo to the dev repo.
git pull
```

Use an editor to change *file1* and *file2*.

```
# Stage file1 and file2.
git add file1 file2

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"

# Push the dev repo to the GitHub repo.
git push origin main

# (Optionally) check status.
git status
```

Use Case 3: Removing Files

Perform this step repeatedly throughout the semester as required.

Remove *file1* and *file2* from your development repository and your GitHub repository. To do that, in a terminal window on your development computer issue these commands:

```
# cd to the dev repo directory.
cd cos333asgt1

# Pull the GitHub repo to the dev repo.
git pull

# Stage file1 and file2.
git rm file1 file2

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"

# Push the dev repo to the GitHub repo.
git push origin main

# (Optionally) check status.
git status
```

Use Case 4: Resolving Conflicts

Perform this step repeatedly throughout the semester as required.

A *conflict* occurs when (1) your teammate pushes a new version of a file to your GitHub repository, (2) you edit an old version of the same file in your development repository, and (3) you attempt to pull from the GitHub repository to your development repository or push from your development repository to the GitHub repository.

This sequence of commands illustrates a conflict and how to resolve it...

Your teammate issues these commands:

```
# cd to the dev repo directory.
cd cos333asgt1

# Pull the GitHub repo to the dev repo.
git pull
```

You issue these commands:

```
# cd to the dev repo directory.
cd repodir

# Pull the GitHub repo to the dev repo.
git pull
```

Your teammate uses an editor to change *file1*. Your teammate issues these commands:

```
# Stage file1.
git add file1

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"
```

You use an editor to change *file1*. You issue these commands:

```
# Stage file1.
git add file1

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"
```

Your teammate issues this command:

```
# Push the dev repo to the GitHub repo.
git push origin main
```

You issue this command:

```
# Push the dev repo to the GitHub repo; fails!
git push origin main
```

Your `git push` command fails because of a conflict. Git recommends that you pull from the GitHub repository.

So you issue this command:

```
# Pull the GitHub repo to your dev repo.
git pull
```

Git notes that *file1* is in conflict. Git annotates *file1* to note the points of conflict. You edit *file1* to resolve the conflicts and eliminate the annotations. Then you issue these commands:

```
# Stage file1
git add file1

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"

# Push the dev repo to the GitHub repo; succeeds!
git push origin main
```

Use Case 5: Branching

Perform this step repeatedly throughout the semester as required.

You decide to implement a new experimental feature in your application. It may not work out, so you don't want to implement the experiment in the *main* branch. Instead you decide to implement the experiment in a new branch named *exp*.

You issue these commands:

```
# cd to the dev repo directory.
cd cos333asgt1

# Pull the GitHub repo to your dev repo.
git pull

# Create a new branch named exp.
git branch exp

# Make exp the current branch.
git checkout exp
```

You edit *file1* extensively, but not completely. Oh no! Your teammate informs you of a bug in *file1* that you must fix in a hurry. It's a good thing that you're implementing your experiment in a non-main branch! You issue these commands:

```
# Stage file1.
git add file1

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"

# Make main the current branch.
git checkout main
```

You edit *file1* to fix the bug. Then you issue these commands:

```
# Stage file1.
git add file1

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"

# Push the main branch to the origin (GitHub) repo.
git push origin main
```

You decide to resume your work on the experiment. Of course you must merge your bug fix into your *exp* branch. You issue these commands:

```
# Make exp the current branch.
git checkout exp

# Merge the main branch into the current (exp) branch.
git merge main
```

Git notes that *file1* is in conflict. Git annotates *file1* to note the points of conflict. You edit *file1* to resolve the conflicts and eliminate the annotations. Then you issue these commands:

```
# Stage file1.
git add file1

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"
```

You finish editing *file1* with your experimental code. Then you issue these commands:

```
# Stage file1.
git add file1

# Commit the staged files to the dev repo.
git commit -m "Message describing the commit"
```

You're happy with the outcome of the experiment, so you decide to merge your *exp* branch into the main branch. You issue these commands:

```
# Make main the current branch.
git checkout main

# Merge the exp branch into the current (main) branch.
git merge exp

# Push the main branch to the origin (GitHub) repo.
git push origin main
```

You decide that you no longer need your *exp* branch, so you issue this command:

```
# Delete the exp branch from your dev repo.
git branch -d exp
```

Recommendations

Use branches to develop new features!

Try to avoid conflicts! To do that, (1) pull from your GitHub repository before starting each programming task, and (2) push to your GitHub repository often – as soon as the programming task is finished to your satisfaction.

When doing an assignment, you or your teammate must copy your assignment code to courselab – for final testing and submission. To copy your assignment code to courselab you could use sftp, maybe via the FileZilla application. However, a more efficient approach is to use Git. You could configure Git on courselab, clone your GitHub repository to a development repository on courselab, and repeatedly pull your code from your GitHub repository to your development repository on courselab as necessary.

Learning More

There is much more to Git. To learn more I recommend that you read the Git book at https://git-scm.com/book/en/v2, and the Git reference manual at https://git-scm.com/docs as required.

This document describes how to use Git through its command-line interface. An alternative is to use a graphical user interface. The page https://en.wikipedia.org/wiki/Comparison_of_Git_GUIs lists many Git graphical clients. Former COS 333 students have recommended *GitHub desktop* (for Mac and MS Windows) and *TortoiseGit* (for MS Windows).

There also is much more to GitHub. I recommend that you investigate these features after you're comfortable with GitHub basics:

- **Projects**: for tracking goals and progress of a general "project". The official documentation is at https://help.github.com/en/github/managing-your-work-on-github/about-project-boards. A reasonable video tutorial is at https://youtu.be/ff5cBkPg-bQ.
- **Issues and Pull Requests**: for defining and implementing small program features in a collaborative and scalable way. An introductory guide is at https://guides.github.com/activities/hello-world/. A verbose set of documents about GitHub *flow* is at https://help.github.com/en/github/collaborating-with-issues-and-pull-requests.
- **GitHub Pages**: for website hosting directly from source code. GitHub's own Pages introduction is at https://pages.github.com/.
- **Actions**: for automatically running code like linters, builders, and deployers whenever you push to GitHub. The official documentation is at https://help.github.com/en/actions/automating-your-workflow-with-github-actions.
- **Tags and Releases**: for creating *versions* of your code both to release and reference. *Tagging* in Git is covered at https://git-scm.com/book/en/v2/Git-Basics-Tagging). Creating GitHub *releases* is covered at https://help.github.com/en/github/administering-a-repository/creating-releases.