

**Project 1.3.9 Tools for Collaboration**

Introduction

|  |  |
| --- | --- |
| Version control is a powerful tool for developing any project or product. If you are collaborating with one or more other people, it is even more important. Have you ever found it difficult to keep papers well organized? What happens when 20 people all work on the same document? | GitHub.com/about  The GitHub Team |

Materials

 Computer with Enthought Canopy distribution of *Python*® programming language, Internet browser, and GitHub client software

 GitHub individual account and membership in organizational account's team

**Procedure**

**Part I: Background Information**

Frequent changes and testing are important during development of a digital product. A **version control** system allows a person to create software or any other product and keep track of those changes. A **distributed version control** system allows a group of people to work on the same project and keep track of everyone’s individual changes.

There are many different distributed version control systems, including BitKeeper, Mercurial and git. Git was developed by Linus Torvalds for use during development of the Linux operating system kernel, and has since become the most widely used distributed version control software.

Git works by tracking a **repository**, which is a folder containing subfolders with files for a project. The files can be code, images, sound, or anything else. A developer can change the files and save them just like usual with the computer's operating system. When the developer wants to record a group of changes, they record it with a **commit**. A commit is a checkpoint in the development of a project.

**GitHub** is an online git repository hosting service that makes it easy to collaborate on a project with multiple developers. GitHub consists of a web portal to your online repos and a local client program that can access both your online repos and your local repos. The local client lets you use git with either a graphical user interface or a command line interface.

**Part II: Workflow**

There are three distinct parts to the software you will be using: The git shell (the command line for git), GitHub for Windows Client (the GUI for git on your computer) and the GitHub web interface (online at [github.com](http://github.com/)). This document will indicate what you need for each step with the icons shown below:

|  |  |  |
| --- | --- | --- |
| Shell | Client | Web |
| This symbol means  use the git shell | This symbol means  use the GitHub-For-Mac/Windows Client | This symbol means  use the GitHub  web browser interface |
| Code Editor | IPython | OS Explorer |
| This symbol means  use the code editor  in Canopy | This symbol means  use the IPython session  in Canopy | This symbol means  find the file in folders in the operating system |

GitHub hosts many different repositories for projects, many of them **open source**. Open source projects are public and make their source code available to anyone, usually free of charge, and many people often contribute modifications and enhancements to the project. Most repositories on GitHub are shared as read-only, meaning others can view and copy the project, but cannot modify it without the approval of the owner or manager.

When a developer makes a copy of a shared read-only repository, they create a **fork**. (Think of a tree in the park: the main trunk is the original repository, and it can split off into another direction at a fork.) A developer can then make changes to their copy, and if they change something that they think should be incorporated in the main project, they can make a **pull request**, asking the project owner to “pull” their changes and **merge** them into the original code.

A repository is always kept both on the developer’s computer (**local**) and online on the GitHub servers (**remote**). Once a repository is forked, it is then cloned to the local computer so the code can be worked on. As changes and commits are made, the local repository is synced with the remote repository to keep the two copies “in sync” with each other.

A repository can also have multiple strands of development going on simultaneously. These are called **branches**. (Think of a tree in the park again: a trunk or fork of a trunk can split off into multiple branches.) A repository can be forked from another repository, and each repository has a master branch, and can have any number of other branches. Often times there are branches for things like bug fixes, testing, and enhancements. At any time these branches can be merged back into the master branch, just like a fork can be merged back into the master.

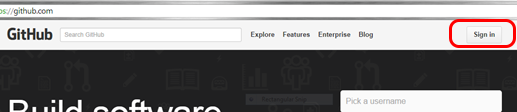
Now it’s time to put all this to work!

**Part III. Setup**

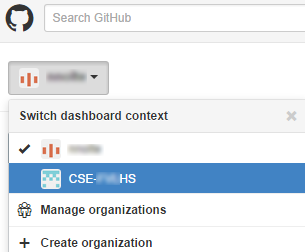
Your teacher has created a repository for your class and shared it with you through the teacher's organizational account on GitHub. You need to **fork** it, **clone** it, and make a **branch** for testing.

1.  Create a fork of the repository.

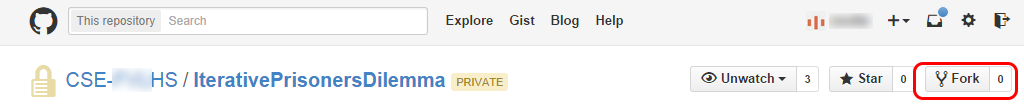
a. Go to [github.com](http://github.com/) and login to your account.



b. Switch to your teacher's organizational account.

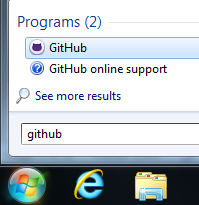


c. Click on the IterativePrisonersDilemma repository and click “Fork” to make a copy of it in your account.



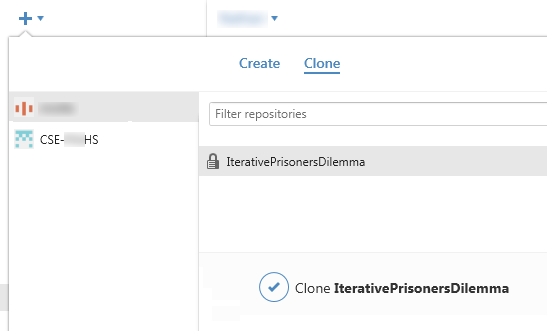
2.  Clone the repository to your computer as follows.

a. Open the GitHub for Windows Client on your computer.

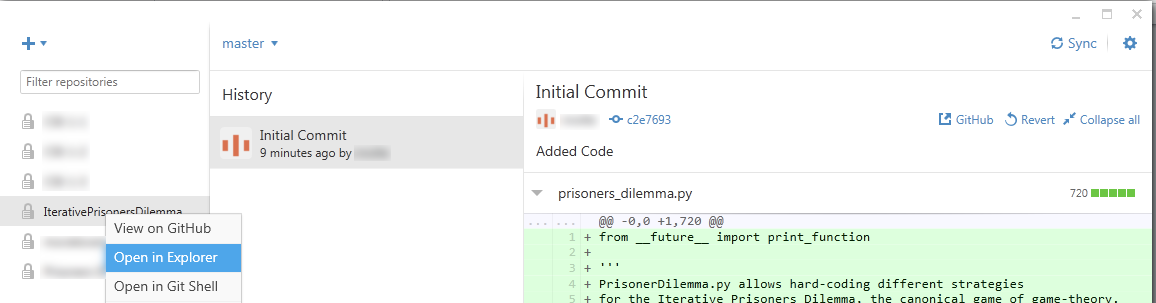


b. Go to Tools 🡪 Options and log in to your GitHub account.

c. Click on the “+” symbol, choose “Clone”, click on the IterativePrisonersDilemma repository and click the checkmark at the bottom. Choose a location to save the repository as directed by your teacher.



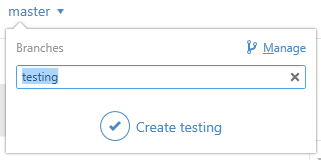
d. Once the repository is cloned, you will see it in the list on the left of the window under the “+”. Left-clicking on it opens it in the client. Right-clicking on it gives you several options, including opening it in Explorer. The history probably only has one commit, because that is when your teacher published the original repository. Right click to open the repository in Explorer.



e.  You should be able to find the prisoners\_dilemma.py file in an OS Explorer window. You may also see a .git folder. This is where important information is kept that git uses to keep track of changes. Don’t make any changes to the files in this folder.

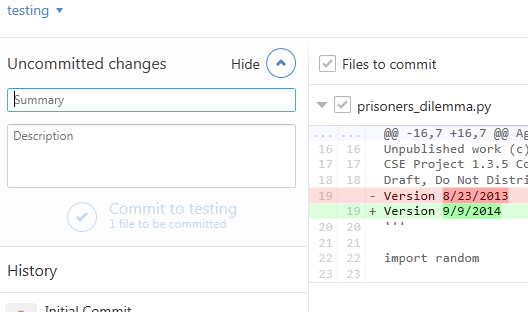
3. Make a “Testing” branch and a commit as follows.

a.  Back in the GitHub For Windows Client window, note the word “master” with an arrow next to it. That indicates you are working on the master branch. The master branch is what we will eventually make our changes to in order to submit a pull request to merge our code with the rest of the class. For now, we want a “testingAB” branch where "AB" are your initials. This branch will allow us to explore and make changes to the code without affecting the master branch. Click on the arrow, type “testingXX” in the box, and click the checkmark to create a new branch.



b. Note that it now says “testingXX” where before it said “master.” That means you are on the testingxx branch, not the master branch. The prisoners\_dilemma.py file is in the folder for your local repository. Open the *Python* file with Canopy. Change the date in the *Python* file at the end of the multiline comment. Save the *Python* file.

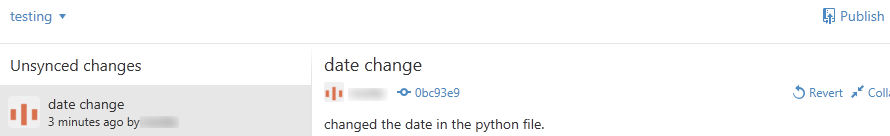
c. Go back to GitHub For Windows. It should now show that there are uncommitted changes to the repository. Click on the arrow next to uncommitted changes, and it will show you a preview of the file with the date change you made.



d. In order for git to make a checkpoint of this change, you need to make a commit. Enter a summary (“Date Change”) and description (“Changed the date in the *Python* file.”) and click “Commit to testingxx.”

e. You now have a record of that change in the testingxx branch. However, the master branch is still unchanged. Click on the arrow next to testingxx and switch back to the master branch. You will see that there are no changes to that branch. Also, when you go back to *Python*, you will see that the change you made has disappeared! Git actually changed the file on the computer to reflect what is current in the master branch. The change still exists – try changing back to the testingxx branch. You can see how you can make changes to code without affecting the master version of the code and easily switch between the two.

f. One last step – you need to sync your local changes to the remote repository on GitHub. Since you have not synced since you created your new branch, it says “Publish” in the upper right hand corner. Click on this to send your new branch and commit up to the remote repository.



4. To compete in the tournament, open the document 1.3.9.Pa SupplementIterativePrisonersDilemma.docx and explore the code and make changes. Don’t forget to ensure you are working in the testingxx branch, and make commits and sync as you work. After the tournament, complete the conclusion questions.

Conclusion

1. What is version control?

2. Why is version control so helpful when collaborating with people?

3. How has the Internet changed the way in which people write software – even software not meant for use related to the Internet? Advocates of the Scrum process still recommend that a team of developers be located in the same place to collaborate. Why do you think face-to-face collaboration is still considered superior?