Getting Started with code.seas

How to use the SEAS Code Repository to manage your own work, and to collaborate with others.

Table of Contents:

- Getting Access
  - For SEAS or FAS Account Holders
- Setting Up Your Repository Account
  - Setting Up an SSH key
- Creating an SSH Keypair
- Registering the Public Key
- Creating a Project
- Creating a Repository
- A First Pushing of Code
  - Using Your Private SSH Key
  - An Initial Local Repository
  - Pushing Code
- Cloning Repositories Locally and On the Server
  - Cloning Repositories Locally
  - Cloning Repositories Remotely
- Merge Requests
- Next Steps

Getting Access

The SEAS Code Repository is available for use to holders of FAS or SEAS login accounts; those without such accounts – such as non-Harvard collaborators or those from other Schools – can be granted access by special permission. Please contact us at achelp@seas.harvard.edu for more information.

To begin, you'll need to visit

- http://code.seas.harvard.edu

and will be presented with the basic front page:

The SEAS Code Repository is provided to the SEAS and wider Harvard community for course support, research collaboration, and to make it simple to share your ideas and work with others.

which will show activity of publicly available projects and repositories. Click "Login" from the upper right-hand corner.

For SEAS or FAS Account Holders
If you have a SEAS or FAS username (the part of your email address before @seas.harvard.edu or @fas.harvard.edu), enter it in the User ID field. If you do not have a SEAS or FAS account, contact help@seas.harvard.edu and we can create an account for you.

Note that if you have both a SEAS and FAS account with the same username, the SEAS account will take precedence. Thus if you've forgotten your SAES password, be sure to reset it by visiting the SEAS IT helpdesk or emailing help@seas.harvard.edu.

You will be forwarded to the SEAS OpenID provider page; enter your username and password here, login, and your account is ready to go.

Setting Up Your Repository Account
Once you have an account, you'll be taken to your "dashboard" which shows the status of your projects and repositories, and projects or repositories that you are collaborating on, and of work that you've chosen to "watch." To begin, this will fairly minimal.

To get started you'll need to do four steps at a minimum:

1. Register an SSH key
2. Create a project
3. Create an empty repository
4. Push your code to that repository

Setting Up an SSH key

The main transport for writing data from your local system to the repository is by using GIT over the SSH protocol. As such, in order to write anything you'll need to setup SSH keys so that the repository knows who you are, and trusts you.

Creating an SSH Keypair

To start, you'll need to create an SSH keypair on your local system; this in turn requires that you have SSH tools installed. On Mac and Linux, these tools will already be available; on Windows, you'll need to install extra software. For details on using SSH, see SSH Access to SEAS Hosts. Here we assume that you are familiar with the basics of SSH.

At a minimum you'll need to create an SSH keypair, to be used for git; this can be a dedicated keypair, or in general this step looks something like this on Mac or Linux:

```
$ ssh-keygen -f gitkey
ssh-keygen -f gitkey
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in gitkey.
Your public key has been saved in gitkey.pub.
The key fingerprint is:
$ 
```
In this process, a private key, named "gitkey", and public key, named "gitkey.pub" were created. The private key will be used when we connect to the server using the GIT protocol over SSH; the public key we need to register with the service.

**Registering the Public Key**

To register the public key, we'll copy and paste it into the web GUI. From a browser window where you've logged into the SEAS Code Repository, click "Dashboard" from the link at the top of the page. You'll be taken to your dashboard:

![Dashboard](image)

Click on the button "Manage SSH Keys", and then on the next page "Add SSH Key"

![Add SSH key](image)

You'll be prompted to add the text of the **public** key; the filename of that should which should end in .pub (which in our case was gitkey.pub).

**Add a new public SSH key**

![Add a new public SSH key](image)

Your public key

It is generally located in `~/.ssh/id_rsa.pub` or `~/.ssh/id_rsa.pub`. If you want to use multiple keys you will have

The key should be in the format of:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQAwgBELgYIgcQG5yOQ==
```

Copy the contents of that file into the textbox, and click “Save.” That key will shortly be available for pushing data into the repository.
Creating a Project

Now that you can move data in and out of the system, you need to create a project. Every repository needs to be part of some project, though projects can host more than one repository.

Again, visit the dashboard, and then under the list of projects on the right, click "Create a New Project". You be sent to page where you define the specifics of your project. Since one project can have multiple collaborators and repositories, choose a scale for the project appropriate with a project you are currently involved with, and not too fine grained.

As you enter a title, the "slug" string, to be used in URLs and such will be created for you. Enter any search labels you'd like, choose a license, and enter in URLs if appropriate.

If you'd like to use the Code Repository to provide some basic documentation, you can enable a wiki. Then choose the privacy level for the project; this will apply to all repositories that you create within the project as well. Lastly, add a description blurb for others and for yourself. Then hit "Create Project." It will now be listed on your dashboard, and available for tracking changes. But of course, there's no data there yet! Let's fix that.
Creating a Repository

Once the project is created, you'll immediately be prompted to create a repository:

Add new repository to Simulating Nasal Hair Growth

Here enter the desired name of the repository like any other source code system. Note that this name can't contain spaces or certain other special characters. Also, here indicate the privacy level of this repository and provide a little description. In addition, define whether or not this repository should allow for merge requests. Merge requests let others clone your repository, make changes against it, and then request that those changes be integrated into the main code base. This is a great way to get others to contribute fixes without granting them write access to your repositories, but may not fit your needs. Lastly, hit the "Create repository" button.

New repository created

This repository is being created, it will be ready pretty soon...

If this message persists beyond what is reasonable, feel free to contact us

You'll then be prompted with a “waiting” page while the system creates the repository in the background. This shouldn't take more than 5 seconds.
Once the repository is created, the service will prompt you with instructions for doing an initial push of code or data into the repository, with specific directions for using git.

**A First Pushing of Code**

While the repository is now there, it's empty, so we need to add content to it. In general, the instructions provided are adequate for getting started; the only tricky part may be getting SSH to use your SSH keys.

**Using Your Private SSH Key**

Unless you named the keys you registered the default `id_rsa` or `id_dsa` names, you'll need to tell SSH and GIT to use the keys you've created.

To do so, the easiest solution is to load them into your SSH agent, as described on the page [SSH Access to SEAS Hosts](#). On Windows this might involve using "pageant", on Mac and Linux this usually just entails issuing this command

```
$ ssh-add ./gitkey
```

where you've supplied the path to the private key you created earlier. To test whether SSH is ready to use that key, check with the command

```
$ ssh-add -l
```

If the `gitkey` filename is listed, you should be good to go. If you have troubles with this step, contact us at [achelp@seas.harvard.edu](mailto:achelp@seas.harvard.edu) or refer to the myriad other documentation on the web.
An Initial Local Repository

Once that's working, you can push some code into the repository. If you have a local repository of code ready to go, then great, otherwise start out by creating a dummy repository with an empty README file or similar. Then create a local repository using GIT that includes that file. From the command-line version of GIT, this would be something like

```
$ git init-db
$ echo "Just starting..." > README
$ git add README
$ git commit README
```

See our documentation “Introduction To GIT” for more about the GIT tool itself.

Pushing Code

Once you have a local repository to push, we need to configure it to push to the remote repository on the server, and then actually push it. The “Getting started” snippet on the code.seas.harvard.edu website will detail how to do this for your case. Here we’ll follow the steps for our example case.

To start, you need to configure the local repository with your name and email. From the directory where you created the repository (on Mac or Linux there should be a directory named .git in it), you run

```
$ git config --global user.name "Example User"
$ git config --global user.email "grid@seas.harvard.edu"
```

This tells git who is doing the work locally, for reference when changes are pushed.

Next you define the “remote” repository to push to, in this case named the default, “origin”:

```
$ git checkout master
$ git remote add origin
  git@code.seas.harvard.edu:simulating-nasal-hair-growth/growth.git
```

The “git checkout master” makes sure that you are pushing up the changes to the main trunk of the repository, and not some branch. The next line defines the URL for a remote repository for this local git repository, and need only be done once. The URL is the same string as is shown under the “SSH” option on the page for this repository on the SEAS Code Repository site.

Lastly, now git knows who we are and where we want to push the changes, we can actually upload the changes:

```
$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 239 bytes, done.
remote: => Syncing Gitorious... [OK]
To git@code.seas.harvard.edu:simulating-nasal-hair-growth/growth.git
  * [new branch]      master -> master
$`

Once successfully pushed, the page on the SEAS Code Repository for this repository will show a commit line in the activity, and you'll be able to start browsing the source and commit logs.

This push the local “master” branch (i.e. the main branch) to the remote repository named “master.” In the future, the push can be shortened to just `git push`. 
Cloning Repositories Locally and On the Server

Once your account is setup, and you can start interacting with your own, and with other repositories, and getting down to work. Here’s how.

Cloning Repositories Locally

If you have read access to a repository (i.e. if you can see it), then you can copy or “clone” it to your local system (i.e. workstation, desktop or laptop). Browse to that repository using the web interface and then take note of the URLs presented there.

```
$ git clone git://code.seas.harvard.edu/simulating-nasal-hair-growth/growth.git
$ git clone http://code.seas.harvard.edu/git/simulating-nasal-hair-growth/growth.git
$ git clone git@code.seas.harvard.edu:simulating-nasal-hair-growth.git
```

For the last clone command, you need to ensure that your SSH keys are setup properly as before, since, as noted above, the protocol is authenticated.

Once cloned to your local machine, you can make changes, commit them, and then push those changes back to the server, whether back to this remote repository (you’ll need to have authorization for that of course) or to a new repository. Remember that all “pushes” (i.e. writes) back to the server must use the SSH transport protocol.

Cloning Repositories Remotely

One advantage of the SEAS Code Repository (and other services like Github or Gitorious) is that you can clone a repository remotely, without pulling down the data to your local machine. This makes a nice mechanism for students to checkout homework, or for people to experiment with a codebase without touching it directly.

Create a clone of growth in Simulating Nasal Hair Growth

Owner

Me

Name (eg "example-user-sandbox", "performance-fixed etc"): growth-cloned

Clone repository
To do this, you simply click the “Clone repository” button (see image above) and then give it the clone a new name. You'll then see a “waiting” dialogue, and then a clone repository that you own and is private to you. Note that the clone repository is still in the original project, but is owned by you now, even if the original repository was not.

**Merge Requests**

You can then clone this locally to your client machine and develop against this codebase. Once changes are in place, you can request that your changes be incorporated into the original code by creating a “merge request”.

Click the button “Request merge” and then fill out the ensuing form, in particular noting that you want to merge with the original repository that you cloned from.
Create a merge request

By sending a merge request to another repository you are inviting the owners of that repository to merge the changes you have in your clone. By writing a short summary explaining your overall changes, the recipients can decide whether to merge your commits or provide feedback on them. The merge requests are public so that everyone can provide feedback on your changes.

Summary
A one-line summary of your changes

Description
A more detailed overview of your changes

Target Repository
The one you wish this repository should be merged with

Target Branch
The target branch you wish your changes to be merged into

Source Branch
The source branch you wish the target repository should merge from

Select commits:
Select the range of commits you want to be included in the merge request

Create merge request

Once the merge request is completed, the owner of the original repository will be alerted, can view the changes from the original, and can agree to the changes or reject them.

Next Steps

For more features, see the followup document

- Advanced Features of code.seas