

# Git basics

Git is an invaluable tool to programmers and computational linguists alike. Version control is a powerful tool that can make integration with large groups of people a breeze and allow one to rectify regressions promptly. This is not a very intricate tutorial for how git works under the hood (though there is a small bit of discussion about this). The primary focus is in how to use git effectively.

- [Cloning a Repository](#)
- [Git hist](#)
- [Introduction](#)
- [Checkout](#)

# Cloning a Repository

This chapter discusses how to clone a repository. This is most likely the first thing one will do with a git repo. With modern git platforms like github, bitbucket, gitea and gitlab, initialization of repositories is not typically done.

Instead, the repo is initialized on the server managed by one of these git platforms and then the repository is cloned onto your local machine.

## To clone a repository

There are two protocols used to clone a repository: ssh and https. Both are secure and encrypted in their transfer and both are rather quick. I prefer to use ssh as my git workflow is improved with the use of this protocol. This is because, particularly with the lengthy passphrases required by Indiana University, the IU Github instance is a pain to pull and push from. By using ssh, my key can have whatever password I want or no password at all while still maintaining a high level of security.

Here are some of the primary differences between the two protocols with regard to how one interacts with them in git. You can switch the protocol used after the initial clone. However, it is simpler to clone using the protocol that you prefer right from the start.

## SSH

More initial setup is required to use ssh with git. The urls for cloning using ssh typically look something like this

```
git@bitbucket.org:ksteimel/test_project.git
```

Let's break this down really quick. The beginning `git@` part says that this ssh protocol is actually

using the git user on the server. In order for this to work, the server needs to have a preshared key from our local machine.

To generate this key and put it on the server, follow the following instructions on a mac or linux machine:

1. Check to see if there is a file in `~/.ssh/` that ends in `.pub`. The default file is normally called `id_rsa.pub` on most modern systems.
2. If this file does exist, and you know the password associated with this key, simply open the file in your favorite text editor, copy the content to your clipboard and paste the contents into a new key in the web interface to the git server. Creating a new key is usually done by accessing your user settings (by clicking on your user icon and then clicking 'Settings') and then going to the submenu dealing with SSH & GPG keys.
3. If this file does not exist, create the file
  1. Run the command `ssh-keygen` on your local machine as the user you would like to use for this repo.
  2. This program will prompt you for the location where you would like to store the keys as well as the password for the keys. Typically, I leave these both at their defaults on OpenSuse which is `~/.ssh/id_rsa.pub` for the location and nothing for the password.
  3. If you have a different key location/name, you will need to add some content to your ssh config file. In `~/.ssh/config` put the following:

```
Host github.com
IdentityFile <path to identify file>
User git
```

4. Never upload your private key. This will not work and it will also leave your system exposed. Private keys are to be guarded closely, public keys are to share around.
4. After the public key file has been created, you should add this public key to the git server's web management system using the instructions in 2

# HTTPS

To use https, simply select the https url option when cloning. There is no additional setup required. However, every time you push or pull, you will have to enter your username and password. If you use the credential helper, this can allow you to avoid this annoyance. To do this, run the following `git config credential.helper store`. You should only be prompted for your username and password one more time and then git should remember. If you need to change your credentials for any reason, just rerun that command and then you'll be prompted for credentials the next time you pull.

# Git hist

This is an excellent alias for git hist to add to your .gitconfig file

```
git log --pretty=format:"%h %ad | %S%d [%an]" --graph --date=short
```

[This page](#) has an excellent discussion of how to add this alias to your .gitconfig file. This website is also the source of this handy alias.

The result is a description of the repository's history with color charts showing the different branches. It's all done in the terminal too!

This is an example of what the hist command produces.

```
* 941114d 2018-08-15 | Added reference to resources page in index page [Kenneth Steimel]
* 5efaf1a 2018-08-15 | Added resources file with helpful templates as well as past conference presentations
* f2f1cc8 2018-08-15 | Added zip'ed version of tex template for presentations [Kenneth Steimel]
*   edf1263 2018-08-11 | Merge branch 'master' of ssh://ksteimel.duckdns.org:332/ksteimel/bio [Kenneth Steimel]
| \
| * c88d193 2018-08-11 | fixed errors in the default layout linking back to main page [ksteimel]
| * 2a83704 2018-08-11 | Added site url to the ruby config file [ksteimel]
| * | 0543da8 2018-08-11 | Added moth video [Kenneth Steimel]
| /
* 2f93f49 2018-08-11 | Should have used baseurl instead of url to take me to the main site. Fixed now [Kenneth Steimel]
* 911b4e9 2018-08-11 | Made the website title a link back to the homepage [Kenneth Steimel]
* fe5a6cd 2018-08-11 | Added roast notes for honduran shg santa rosa [Kenneth Steimel]
* 4238b53 2018-08-11 | Added image of roaster and linked to image of roaster in blog post [Kenneth Steimel]
* 032f806 2018-08-11 | Added roasting procedure I use [ksteimel]
*   bec9142 2018-08-11 | Merge branch 'master' of ssh://ksteimel.duckdns.org:332/ksteimel/bio [Kenneth Steimel]
| \
| * 7c034b9 2018-08-11 | Changed [ksteimel]
| * | 6d1b477 2018-08-11 | Added page title to a header element in the body of the page [Kenneth Steimel]
| /
* d54fd17 2018-08-11 | Changed background color to a more periwinkle color [Kenneth Steimel]
```

# Introduction

[The Guided Git Tutorial](#) is an invaluable resource for learning how to effectively use git. This chapter is similar and draws from this tutorial in many ways. However, the focus here is to provide a more step by step explanation of how to use the pieces of git that are important for computational linguists to know how to use.

Yue Chen and I have also prepared other materials to assist with learning git as well. Namely the presentation in the sidebar of this page. In addition, I have a [git repository](#) that walks you through many of the topics discussed in this chapter.

# Checkout

One of the most powerful functionalities provided by git is the checkout functionality.

This command can be used to do a number of different things including:

- Creating a new branch
- Switching to a different branch
- Updating the working tree
- Moving to a different commit on the current branch
- Creating a new branch at a previous commit point

I'll go over each of these functionalities and provide a use case for them as well.

## Creating a new branch

This can be useful if there are multiple people working on a project where there are subgroups in the project that are working on different parameter settings or different feature extraction methods. Rather than create two different directories where the files live, you can create two separate branches and use the checkout command to switch between them.

To create a new branch use the following syntax

```
git checkout -b <new branch name>
```

## Switching to a different branch

Now that this new branch is created though, how do you quickly switch back and forth between them? The answer is to essentially leave out the `-b` flag.

```
git checkout <target branch name>
```

## Updating the working tree

The most common case where I use this functionality is to get back a file that I accidentally deleted from my file system. Running `git checkout <file path>` will bring the latest version of that file into your repo, even if that file has been deleted by mistake.

## Moving to a different commit on the current branch

If there's a previous commit that you would like to roll back to, for example, if you need to examine the previous way that some system was running, you can checkout an individual commit from the past. To do this, you will need to obtain the shortened hexadecimal commit hash. One way to obtain this is using the `git log` command. I recommend using the `git hist` alias discussed elsewhere in this chapter.

Another way is to look at the repository in the git web interface on the server where your repository is hosted. There will typically be a place to view the commit history on these web interfaces. Then you can copy the commit code and paste it into the appropriate spot in your command line.

For example,

```
git checkout 79b47a4
```

Where `79b47a4` is an example commit code.

## Creating a new branch at the previous commit point

However, the previous command will put your local repository into a detached head state. You can do most git things but you are not currently on a branch terminal so some operations like merging will not work. The solution is to create a new branch when you rewind to a previous commit. This essentially combines the methods from two of the previous sections. Simply run

```
git checkout -b <new branch name> <commit hash>
```