

# Practical Tools for Computational Linguistics

This book is a work in progress. It details pieces of tech that are necessary for Computational Linguistics and Natural Language Processing. This is primarily created for fellow students at my school (Indiana University - Bloomington). Though I've made this publically available. If someone else finds this useful, send me an email at [ksteimel@iu.edu](mailto:ksteimel@iu.edu).

- [Git basics](#)
  - [Cloning a Repository](#)
  - [Git hist](#)
  - [Introduction](#)
  - [Checkout](#)
- [Supercomputer tools](#)
  - [Containers on IU supercomputers](#)
  - [Supercomputer info](#)
  - [Notes on Screen](#)
  - [AMD optimized crfsuite](#)
- [Organizational Information](#)
  - [ClingDing Spring 2019](#)
  - [CLINGDING Fall 2019](#)
- [julia](#)
  - [Graphing](#)
- [Python notes](#)

- Docstring example
- CRF-suite on AVX2 cpus
- AllenNLP
  - Notes

# 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

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 iamge 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>
```

# Supercomputer tools

This describes some information about the super computer tools that are available at IU as well as information on how to use these tools.

# Containers on IU supercomputers

Karst, Carbonate and Big Red II have the singularity package available which allows users to run docker or singularity images on the supercomputers. This can be a great way to run programs that are not installed on the supercomputers.

To use singularity, load the module

```
module add singularity
```

Then pull down an image. You can pull an image from dockerhub or singularity's hub.

```
singularity pull docker://julia:latest
```

Unlike docker, singularity creates a file that contains the image specification. To run the container use the image file generated by your pull command.

```
# singularity exec <local image> <cmd>  
singularity exec julia-latest.simg julia
```

For more information, please see the [singularity documentation](#).

Supercomputer tools

# Supercomputer info

Carbonate node: 710.223 GFLOPS on intel mkl linpack

If you need to get 32GB of VRAM on Carbonate-dl

From a previous help ticket: "dl[11,12] are in fact v100-16GB parts. If the user needs v100-32GB they'll need to add a "-w <node>", where <node> is dl1 or dl2."

# Notes on Screen

## It is a good idea to use screen sessions

On super computers like those at IU, it is essential to use a screen session for submitting interactive jobs. Screen basically emulates a user sitting at the computer screen. It accepts output and can give input. However, you can detach from screen sessions and log out from the computer without causing running jobs to terminate.

Here are some notes  
provided by IU's [Knowledge  
Base](#)

# When you can't re-attach to your screen session after a lost connection

In some cases, your previous screen session may not have detached properly when you lost your connection. If this happens, you can detach your session manually.

To see your existing screen sessions, enter:

```
screen -list
```

This will display a list of your current screen sessions. For instance, if you had one attached and one dead screen, you would see:

```
There are screens on: 25542.pts-28.hostname (Dead ???) 1636.pts-21.hostname (Attached)
Remove dead screens with 'screen -wipe'. 2 Sockets in /tmp/screens/S-username.
```

To detach an attached screen, enter:

```
screen -D
```

If you have more than one attached screen, you can specify a particular screen to detach. For example, to detach the screen in the above example, you would enter:

```
screen -D 1636.pts-21.hostname
```

Once you've done this, you can resume the screen by entering the `screen -r` command.

(In the above example, the dead screen isn't causing problems, but you should probably enter the `screen -wipe` command to get rid of it.)



# AMD optimized crfsuite

## Problem

The bundled binary from python-crfsuite and sklearn-crfsuite performs rather badly on amd processors.

For example, in a trial run with training a small pos tagger on an AMD Epyc 7601, each iteration in the hyperparameter search took about 1 minute 15 seconds. This is only with a small training set of about 400 sentences. With the full 6,000 sentences available it takes upwards of 4 days to finish a single run. Rough.

However, on a dual intel E5-2680 system (16 cores at 3.2 Ghz all core turbo), the performance is much faster. On that same small dataset of 400 sentences, each iteration takes about 30 seconds, the entire hyperparameter search over 50 combinations takes 2 minutes.

This appears to be due to the fact that the binaries that ship with sklearn-crfsuite were compiled on an intel platform.

## Solution

To fix this, I created a fork of python-crfsuite that uses the avx2 instructions available on amd's zen processors (this should also help with newer intel processors that have avx extensions).

This fork is available at <https://github.com/ksteimel/python-crfsuite.git>

To use this, clone the repo

```
git clone --recurse-submodules https://github.com/ksteimel/python-crfsuite.git
```

Then change into the new directory.

```
cd python-crfsuite
```

it's a good idea to create a virtual environment so if you decide you don't want to use this version, you don't have to.

```
virtualenv -p <your python location> <path to virtualenv>source <path to  
virtualenv>/bin/activate
```

Then, we need to build the package.

```
python setup.py build  
python setup.py install
```

Now you should have an optimized build of python-crfsuite You can then install sklearn-crfsuite .

```
pip install sklearn-crfsuite  
pip install scikit-learn
```

To get out of your virtual environment run,

```
deactivate
```

## How did we do?

The whole point of this was to speed up performance on amd processors with crfsuite. If we go back to our small training dataset, we see a substantial boost in performance.

We've now gone from one minute 15 seconds per iteration to only 30ish seconds per iteration.

It may seem somewhat shocking that the amd processor is only about as fast per iteration as the intel processor. However, the amd processor has double the number of cores (32 instead of 16)

with a lower clock speed. (2.2 Ghz for the epyc processor compared to 3.2 for the intel procesor). If we look at the entire grid search across 50 parameters, we see the core advantage of the epyc processor emerge: the grid search finished in only 1.1 minutes on the AMD processor while it took 2.0 minutes on the pair of intel processors.

Not too shabby for 2 minutes of work.

# Organizational Information

# ClingDing Spring 2019

January 23rd	Coffee @ <a href="#">Pourhouse</a>
January 30th	<a href="#">NACLO</a> grading party
February 6th	Coffee
February 13th	git tutorial (Yue Chen & Ken Steimel)
February 20th	Noon -- 1: Mel Andresen 4 -- 5: Coffee
February 27th	Job/Internship hunting
March 6th	Coffee
March 20th	Yue Chen Ken Steimel :: <b>Cross Language Tagging in Luyia</b>
March 27th	Coffee
April 3rd	Yue, Ken, Leah, Noor, Zhouyu :: <b>Abusive Language Detection</b> Hai Hu
April 10th	Coffee
April 17th	Noor Abomokh
April 24th	Automated testing in python (Ken Steimel)
May 1st	Coffee

# CLINGDING Fall 2019

- 9-4: Coffee (crumble 10th and college)
- 9-11: Internship: noor, josephine, ken, hai
- 9-18: Coffee
- 9-25: Yue
- 10-02: Coffee
- 10-9: Nastia
- 10-16: Coffee
- 10-23:
- 10-30: Coffee
- 11-6:
- 11-13: Coffee
- 11-20:
- 11-27: Thanksgiving
- 12-4: Coffee
- 12-11: Becca
- 12-18: Coffee

julia

julia

# Graphing

The GR package seems to be much quicker to get to first graphing than the `pyplots.jl`, `Gadfly.jl` or other packages. However, I do like the way `gadfly` looks better.

`VegaLite` also seems to be a rather quick package



# Python notes

# Docstring example

```
class Albatross(object):    """A bird with a flight speed exceeding that of an unladen
    swallow.

    Attributes:
        flight_speed        The maximum speed that such a bird can attain.
    nesting_grounds    The locale where these birds congregate to reproduce.
    """

    flight_speed = 691
    nesting_grounds = "Throatwarbler Man Grove"
```

# CRF-suite on AVX2 cpus

using sklearn-crfsuite or python-crfsuite on an AMD system can be very slow due to optimizations in the precompiled wheel files that are specific to intel processors.

I have a branch of python-crfsuite that has flags for avx2 instructions. To see if your cpu supports avx2 instructions, examine the output of `lscpu | grep avx2`. If anything is returned, then your cpu supports the avx2 instruction set.

## To use this fork:

### Create a virtual environment for this version of python-crfsuite

```
virtualenv ~/venvs/crfsuite  
source ~/venvs/crfsuite/bin/activate
```

### Clone my fork of python-crfsuite

```
git clone --recurse-submodules git@github.com:ksteimel/python-crfsuite.git
```

### Build python-crfsuite

```
python setup.py build  
python setup.py install
```

# Install additional dependencies

```
pip install sklearn-crfsuite  
pip install scikit-learn
```

## How much does this help?

Even on intel cpus that support avx2 instructions, the time taken to complete a grid search is reduced. For example, a 5 fold grid search with 10 parameter combinations (e.g. 50 total runs) takes 4 minutes to complete using the version in pypi (on a POS tagging problem in a Turkic language). The avx2 compiled version completes in 3 minutes. This becomes more pronounced as the size of the tagset increases.

Using avx only (e.g. changing -mavx2 to -mavx in the setup.py script) still results in improvements to performance. On a pair of intel e5-2680, here are the runtimes for this same benchmark script in minutes. Note that the avx on setting for 16 is having trouble even keeping the cpus loaded because the grid search runs are finishing too fast. This is an issue when the number of tags is small as each task in grid search finishes very quickly and most of the time is spent on task overhead. with longer running tasks, the difference is more noticable.

job	avx off	avx on
8	3.5	2.7
16	2.8	2.5

# AllenNLP

# Notes

If you're running pytest and you have your own modules, sometimes it's necessary to run pytest as a python module.

```
python -m pytest
```

This automatically imports the current directory into your PYTHONPATH.

Just using pytest by itself can cause issues even with `python_paths = ./` in your `pytest.ini`.