Merchant Venturers School of Engineering

Outreach Programme

What Is Git?

And how do I use it?

Created by

Ed Nutting

Organised by

Caroline.Higgins@bristol.ac.uk

Published on July 20, 2016

Notes to the reader

- This workshop is intended to last 1 hour.

- This workshop is intended for people with no prior knowledge of Git, GitHub or GitKraken. You don't need to be able to use the command line for this tutorial (or to use Git!).

- This workshop aims to teach the skills needed for working on small Outreach projects (i.e. other workshops). As such, we will ignore many of the feature of Git that you would need in industry. This is a good starting point though.

- The content is intended to be learnt through self-directed individual learning by following the worksheet and practicing actions repeatedly until you remember how to do them.

- The learning platform is Git for Windows or Linux and GitHub. For both platforms, we will use the GitKraken user interface (sorry, but command line people can probably already figure out how to use Git on their own).

- This workshop probably works on Mac but is untested (unless anyone fancies donating a Mac to me? :) #worthatry)

- There are many of versions of Git, all of which should be compatible with this workshop (but might not be - if you follow this guide, you'll end up with the right one installed.)

- You should already be comfortable using your platform of choice including: installing programs, opening programs, managing files and folders, signing up for stuff via email and accessing the internet (which is presumably how you got this worksheet in the first place?)

- This workshop teaches the following skills:

– Creating an account and repository on Github
– Cloning a new/empty repository
– Configuring a new/empty repository
– Installing a git client
– Configuring a git client
– Adding (/creating) files inside your project
– Pushing and pulling files
– Viewing the commit history
– Correcting a mistake in a file
– Discarding some or all changes to a file
– Viewing, creating and commenting-on issues on GitHub

1 Introduction

Hi! In this short workshop we're going to try to introduce you to Git - industry's (pretty-much) standard tool for version control and what we use in the MVSE Outreach programme for managing files. Don't worry if you have no idea what "Version Control" is just yet - we're going to start from the ground up.

1.1 What is version control?

Let's suppose you create a file - something simple like a Word document. You write an essay, suppose 10,000 words, as a first draft. Somewhere along the line you write a really good paragraph with a great explanation of a concept. You send your essay off for review. When you get it back, its got lots of comments on it. So you save the comments and duly start editing. After a few days, you realise that as part of your editing you've deleted that really good paragraph because it didn't seem relevant but now you want it again - what do you do? Try to remember it and write it again? What a pain that would be...

A lot of people solve this by saving new copies of files as "My File Version 1/2/3/4/etc." or (possibly better) "My File - YYYY-MM-DD hh:mm". This has several issues, namely: it generates a lot of files, it's very hard to compare versions, it's hard to find which the first/last version was that had a certain bit of content, it's hard to manage and maintain, it only takes on wrong save to destroy a version, and so on and so forth. Ultimately, it's an ineffective way of managing lots of versions of a file. It also becomes next to impossible to share all the versions with other people and takes up a huge amount of disk space with lots of duplication.

So, version control is an efficient, effective way of tracking anything from very few to very many versions of very few to very many files managed by one person or many people.

1.2 Why do we need version control software?

To solve all the aforementioned problems and many other problems. Ultimately, version control is about: maintaining a stable latest version, keeping a history of previous versions, being able to recover stuff from previous versions, and being able to share any and all versions with other people.

Most version control also includes a way of comparing versions to see what changed (sometimes call a comparison or "diff" (difference) tool). "diff'ing" two files or folders means using a diff tool to highlight all the differences between the two files/folders. Usually we apply this to two versions of the same file, but, for example, a plagiarism tool might use a diff tool to find similarities between unrelated files.

1.3 So what is Git?

Git is a version control system. It is a system split across multiple computers - so try not to get confused. There is no "one thing" which is Git - several things piece together to make up the complete system.

What makes up the Git system then?

Several parts:

The Respository This is the collective bucket of stuff which contains: all the versions of all the files you are tracking, information about all the people who have ever edited (inc. created, modified or deleted) a file in the repository, and some other information like branches (we won't be looking at branches in this tutorial).

The Server Often called the "Remote Repository" - as the name suggests, it's the copy of the repository ("repo" for short) on the internet that's available to everybody else. The Remote Repo can (essentially) only ever be added to - you cannot delete versions (but you can delete content/files - a new version of something can be a version in which the thing does not exist!).

The Client(s) Often called the "Local Respoistory" - as the name suggests, it's your copy of the repository; the copy on your computer. Nobody else can actually access what's in your local repo. If your hard drive became corrupted, there's no way to retrieve stuff that's in your local repo that hadn't been copied to the remote repo. We'll discuss later how you transfer your local copy of the repo (i.e. your local copy of all the versions) to the remote copy.

What's with the name?

I'll just quote the official answer from Kernel.org Git FAQ : https://git.wiki.kernel.org/index.php/Git

Quoting Linus: "I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'Git'".

('git' is British slang for "pig headed, think they are always correct, argumentative").

Alternatively, in Linus' own words as the inventor of Git: "git" can mean anything, depending on your mood:

- Random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
- Stupid. Contemptible and despicable. Simple. Take your pick from the dictionary of slang.
- "Global information tracker": you're in a good mood, and it actually works for you. Angels sing and light suddenly fills the room.
- "Goddamn idiotic truckload of sh*t": when it breaks

1.4 I've heard a lot about the command line...

Fear not - you do not need to use the command line to use Git. The keyboard warrior CS students will, at this point, spitefully stop reading. But honestly, some very good Git User Interfaces exist and they're better than using command line (because nobody I've ever met has used Git command line without making endless mistakes or simply being less efficient than the GUI versions are). Command line is useful if you want to automate stuff or live under the belief that a mouse is the embodiment of Satan.

1.5 What is a repository (a.k.a. repo)?

I mentioned this above: it's the name for the collection of information being tracked by Git. "Repository" is often shortened to just "repo".

Git is very clever - it doesn't have to know about everything, so you can tell it to ignore files (e.g. temporary Word files). So the Repository refers to anything Git knows about. When Git knows about something, it means it's tracking it. If you create a new version of a tracked file (by editing it in any way, including deleting it), Git will notice and allow you to save that version permanently. "Saving a version" is called "committing" a version - more on this later.

The repo also contains information about people; Specifically, about the people who have created versions. It tracks this so you can compare version and search for who did what and when.

1.6 So what is GitHub?

GitHub is a company who offer a free (for public use) version of the Git Server (the Git Remote Repositories). They are a company that has taken the Git Server software, bought a tonne of hardware and storage space and combined them to give you a place to use for hosting your remote repositories. All public repositories are completely free.

If you want free, private reopositories, take a look at BitBucket. Again, BitBucket is just a company who have taken the Git Server software, added some hardware and offered it for you to use for free.

The point is, GitHub and BitBucket are not "Git" - they are just two of many companies offering you a convenient way of setting up the Server part of the Git System (described previously).

1.7 Remote vs. local versions

We've talked about this a bit already. There's a copy of the repo on the computer you are using (the Local Repo) and there's a copy on the server (the Remote Repo). You save versions to your Local Repo. Then, once you're happy with your most recent version(s), you Push them the the Remopte Repo. "Pushing changes" means copying all the new versions you've saved locally up to the Remote Repo - this makes them available for everyone else to see.

1.8 And what about GitKraken?

GitKraken is a new, recently-out-Beta-testing, pretty-good application for using Git. We'll be using it in this tutorial for managing our Local Repo including pushing changes to the Remote repo.

2 Setup GitHub

2.1 Creating an account

Head to GitHub.com - https://github.com/ and sign up (or log in if you already have an account). If you're going to be working with the University of Bristol Outreach team, please sign up using your Bristol Uni email address (you'll also get access to student perks and discounts this way).

2.2 Joining the MVSE Outreach group

Email Caroline Higgins (or speak to someone already on the group) to be granted access. Please remember to provide the email address you signed up with (which should have been your Bristol Uni email) - it needs to be exactly the same, not one of the many possible variants. You find find who's (publicly visible) in the group here: MVSE Outreach People - https://github.com/orgs/MVSE-Outreach/people. By default, membership is private - it's up to you if you want to be publicly visible.

2.3 Creating a repository

Okay, you've signed up and become a member of the MVSE Outreach group. So that we don't clutter up the Outreach group, we're going to work separately from the group for the rest of the tutorial, but in future just head to the group - https://github.com/MVSE-Outreach to create new repositories within the group.

Actions

1. For now, head to Your Profile - https://github.com/YourUserName e.g. Ed Nutting - https://github.com/EdNutting.
2. Click the Repositories tab at the top
3. Click the green New button (top-right)
4. Enter a name (no spaces) e.g. LearningGit
5. Enter a description if you want to.
6. Select "Public"
7. Check "Initialize this repository with a README"
8. Click "Add .gitignore" and type in "C" (for this demo. In future, use whichever language the majority of your code will be written in for that repo)
9. Click "Add license" then select "None" (Outreach stuff is done totally open source and free)
10. Click "Create Repository"

Licenses

Use no license or a permissive license like The MIT License or The Unlicense when working on Outreach stuff. Preferably don't use any of the GNU or BSD licenses. For you own projects/repos, you're free to use whatever license you like.

2.4 Configuring a repository

Your new repo has now been created. You should have been taken to its main page and been presented with the list of files. If not, head to https://github.com/YourUserName/YourRepoName There's not much to do to configure your new repository. We're not going to touch the settings, just edit a few of the files first. We're going to edit them through the web browser for now, which allows you to make new versions in the Remote Repo directly - you wouldn't normally do this! We'll learn later how to edit files on your computer normally then push

(i.e. upload) them to the remote (i.e. online) version.

.gitignore

This file uses wildcard file names (look that up online) to exclude files from the repo. Specifically, it tells the complete Git system to ignore files names matching a certain pattern thus those files aren't tracked so you can't save versions of them. This is mostly used for ignoring temporary files and build folders.

Actions

1. Click ".gitignore" in the file list
2. Click the "Edit this file" icon in the top-right corner of the file view (looks like a pen)
3. Copy and paste the text from after this box into the bottom of the file on a new line. This will ignore any temporary files created by using LaTeX - you are expected to write worksheets for your workshop using the (very simple and easy to use) LaTeX template found in https://github.com/MVSE-Outreach/LaTeX-Worksheet-Templates/

```
*.tps

## Core latex/pdflatex auxiliary files:
*.aux
*.lof
*.log
*.lot
*.fls
*.out
*.toc
*.fmt
*.fot
*.cb
*.cb2

## Intermediate documents:
*.dvi
*-converted-to.*
# these rules might exclude image files for figures etc.
# *.ps
# *.eps
# *.pdf

## Generated if empty string is given at "Please type another file name for output:"
.pdf

## Bibliography auxiliary files (bibtex/biblatex/biber):
*.bbl
*.bcf
```

```
*.blg
*-blx.aux
*-blx.bib
*.brf
*.run.xml

## Build tool auxiliary files:
*.fdb_latexmk
*.synctex
*.synctex(busy)
*.synctex.gz
*.synctex.gz(busy)
*.pdfsync

## Auxiliary and intermediate files from other packages:
# algorithms
*.alg
*.loa

# achemso
acs-*.bib

# amsthm
*.thm

# beamer
*.nav
*.snm
*.vrb

# cprotect
*.cpt

# fixme
*.lox

#(r)(e)ledmac/(r)(e)ledpar
*.end
*.?end
*.[1-9]
*.[1-9][0-9]
*.[1-9][0-9][0-9]
*.[1-9]R
*.[1-9][0-9]R
*.[1-9][0-9][0-9]R
*.eledsec[1-9]
*.eledsec[1-9]R
*.eledsec[1-9][0-9]
*.eledsec[1-9][0-9]R
*.eledsec[1-9][0-9][0-9]
*.eledsec[1-9][0-9][0-9]R

# glossaries
*.acn
*.acr
*.glg
*.glo
*.gls
*.glsdefs

# gnuplottex
*-gnuplottex-*

# gregoriotex
*.gaux
*.gtex

# hyperref
*.brf

# knitr
*-concordance.tex
# TODO Comment the next line if you want to keep your tikz graphics files
*.tikz
```

```
*-tikzDictionary

# listings
*.lol

# makeidx
*.idx
*.ilg
*.ind
*.ist

# minitoc
*.maf
*.mlf
*.mlt
*.mtc
*.mtc[0-9]
*.mtc[1-9][0-9]

# minted
_minted*
*.pyg

# morewrites
*.mw

# mylatexformat
*.fmt

# nomencl
*.nlo

# sagetex
*.sagetex.sage
*.sagetex.py
*.sagetex.scmd

# scrwfile
*.wrt

# sympy
*.sout
*.sympy
sympy-plots-for-*.tex/

# pdfcomment
*.upa
*.upb

# pythontex
*.pytxcode
pythontex-files-*/

# thmtools
*.loe

# TikZ & PGF
*.dpth
*.md5
*.auxlock

# todonotes
*.tdo

# easy-todo
*.lod

# xindy
*.xdy

# xypic precompiled matrices
*.xyc

# endfloat
```

```
*.ttt
*.fff

# Latexian
TSWLatexianTemp*

## Editors:
# WinEdt
*.bak
*.sav

# Texpad
.texpadtmp

# Kile
*.backup

# KBibTeX
*~[0-9]*
```

Actions

4. Click the green "Commit changes" button

Read-Mes

The Read Me (README.md) file is a Markdown formatted text file which, by default, is displayed beneath the list of files for your repo online. Edit at some point to include information about your workshop Like putting your name on it so you can use it in your CV ;))

3 Setup GitKraken

3.1 Re-cap: What is GitKraken?

TODO

3.2 Installing GitKraken

TODO

3.3 Configuring GitKraken

TODO

4 Use GitKraken

4.1 Cloning a repository

TODO

4.2 Initialising a repository

TODO

Git Ignore: .gitignore

TODO

4.3 Staging files

TODO

4.4 Committing files

TODO

4.5 Pushing commits(/changes)

TODO

4.6 Viewing the commit history

TODO

4.7 Reverting changes

TODO

4.8 Discarding changes before committing

TODO

4.9 Amending commits

TODO

5 Use GitHub

5.1 Viewing your repository

TODO

5.2 Viewing the commit history

TODO

5.3 Viewing issues (Part 1)

TODO

5.4 Creating an issue

TODO

5.5 Viewing issues (Part 2)

TODO

5.6 Commenting on an issue

TODO

5.7 Closing an issue

TODO

6 Wrap-up

We hope you enjoyed this workshop! If you have any questions or run into problems, just ask one of the other outreach people - I'm sure they'll be happy to help.

Goals

Hmmm...

- TODO

7 Extra Resources

Here's a few extra resources to help you along with this worksheet and some stuff to try later.

- Google is your friend : http://www.google.com