Git Tutorials

Read/Watch the required knowledge, and complete each section’s activity

Contents

[1: Setup Git + Source Tree and test working 1](#_Toc114046539)

[2: Create Github User + Helloworld (via GitHub UI) 2](#_Toc114046540)

[3: Add Github Actions to Helloworld (via Git CLI) 3](#_Toc114046541)

[4: Gitignore / Git add: ingest all previous work into Github ( aka Bash / Docker exercises ) 4](#_Toc114046542)

[5: Git Best Practices 5](#_Toc114046543)

[6: Git Merge Hell 6](#_Toc114046544)

[7: Git Hooks 7](#_Toc114046545)

# 1: Setup Git + Source Tree and test working

Follow instructions to install and setup Git  ***Note: if you followed all the Bash tutorial you can skip to next section***  
  
**Required Knowledge:**

none  
 **Activities:**

Install Git-Bash (defacto best git cli for windows)

* <https://gitforwindows.org/>

Install Source Tree (best git repo manager for advanaced git management)

* <https://www.sourcetreeapp.com/>
* <https://community.atlassian.com/t5/Sourcetree-questions/I-always-get-a-popup-credential-helper-selector-asking-me-to/qaq-p/1465880>

Run a version command, to make sure your setup is working

$ git --version

git version 2.31.0.windows.1

# 2: Create Github User + Helloworld (via GitHub UI)

**Required Knowledge:**

None

**Activities:**

1) Sign up for Github with your Riversafe email

<https://github.com/>

2) Create hello world repo in github, following quickstart  
<https://docs.github.com/en/get-started/quickstart/hello-world>

# 3: Add Github Actions to Helloworld (via Git CLI)

**Required Knowledge:**  
Read through sections on Actions  
(aka mini CI for git built into Github)

<https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>

**Activities:**

1) Setup github access using ssh keys

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/>  
<https://docs.github.com/en/get-started/quickstart/set-up-git>

Test access with ssh command (run in bash)

$ ssh -T git@github.com

Text

Description automatically generated

2) Add github action into your helloworld example from the previous section

<https://docs.github.com/en/get-started/quickstart/hello-world>  
<https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions#create-an-example-workflow>

# 4: Gitignore / Git add: ingest all previous work into Github ( aka Bash / Docker exercises )

**Required Knowledge:**

Read through up on git ignore files  
<https://docs.github.com/en/get-started/getting-started-with-git/ignoring-files>  
<https://www.atlassian.com/git/tutorials/saving-changes/gitignore>  
   
*ps excellent gitignore examples* [*https://github.com/github/gitignore*](https://github.com/github/gitignore)  
  
**Activities:**  
Add your previous work into bash, as separate repos  
Make sure to add .gitignore files where appropriate to avoid committing temporary files

# 5: Git Best Practices

**Required Knowledge:**

Read best practice guides  
<https://github.com/thoughtbot/guides/tree/main/git>  
<https://homes.cs.washington.edu/~mernst/advice/version-control.html#best-practices>   
<https://vizer.sharepoint.com/:w:/r/sites/RiverSafe/_layouts/15/Doc.aspx?sourcedoc=%7B572D13FB-2157-47F7-B5CA-A75575FB66D6%7D&file=Git%20Best%20Practice.docx&action=default&mobileredirect=true>

**Activities:**

Review previous git repo in SourceTree and see where best practice could have been applied

# 6: Git Merge Hell

*Task is to solve 4 merge conflict examples, commonly caused when team members don’t communicate together*

**Required Knowledge:**

Read sections on   
<https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/about-merge-conflicts>  
<https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-using-the-command-line>

**Activities:**

Download the zip and presentation of 4 git merge scenarios  
  
Go through presentation, and solve each local git repo’s merge problem  
(goal in each is to merge changes into develop->main or feature->develop)

# 7: Git Hooks

*Note: normally git hook is managed via programming tools like precommit, but it’s important to know how they work under the bonnet*

**Required Knowledge:**  
Read sections on git hooks

<https://www.atlassian.com/git/tutorials/git-hooks>   
<https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks>

**Activities:**

Create a client side “prepare-commit-msg” hook, that fails the commit when the git commit message doesn’t contain the word “please”

Text

Description automatically generated  
  
Hints:

- go to previously create git repo

- in repo create bash script called ‘.git/hooks/prepare-commit-msg’

- make sure script is runnable by git with “chmod +x”  
 $ chmod +x prepare-commit-msg  
  
- update script to fails commits that don’t contain the word “please”  
 hint:

*#!/bin/bash*COMMIT\_FILE="$1"  
COMMIT\_MESSAGE=`cat "${COMMIT\_FILE}"`  
  
echo "file: ${COMMIT\_FILE}, message: ${COMMIT\_MESSAGE}"

Congrats, GIT tutorials are over!