***Git***

PR’s w/ Azure DevOps (Similar to “PR’s w/ Github” below):

First, clone the repository using the https link via the repo on Azure DevOps:

$ git clone https://......

Always pull in local master branch to get the most up-to-date changes, BEFORE creating and working on another branch. This will pull changes from the remote repo on the server, master branch and merge the changes to the local checked-out branch.

$ git pull origin master

Name any new branch you work on with [git branch] t-dad/SOME-NAME

Make changes in the new branch (NEVER master) as normal (i.e., add and commit).

Push the branch onto Azure DevOps (i.e., to the remote repo on the server). This will push all the commits you’ve made in that branch along with it:

$ git push origin t-dad/SOME-NAME

Sanity check via Azure DevOps online to make sure that the correct branch was pushed.

Lastly, make a Pull Request via Azure DevOps online (“New pull request” button under the “Pull requests” tab).

\*\*\*For working with ‘salzburg’ repository (or any repo, really) in VSCode, it is best to close the VSCode window or any processes that might have the repo’s files opened BEFORE SWITCHING TO ANOTHER BRANCH (use Git Bash to perform all the commands)\*\*\*

PR’s w/ Github (Similar to “PR’s w/ Azure DevOps” above):

Before creating a new branch, pull the changes from upstream. Your master needs to be up to date. This will pull changes from the remote repo on the server, master branch and merge the changes to the local checked-out branch.

$ git pull origin master

Create the branch on your local machine and switch in this branch:

$ git checkout -b [name\_of\_your\_new\_branch]

Add/make changes in the new branch. Once ready to commit, use:

$ git add .

$ git commit -m “<message>”

Push the branch on Github. This will push the commits as well.

$ git push origin [name\_of\_your\_new\_branch]

Sanity check on Github before create/merge a new Pull Request.

Delete the local new branch (checkout ‘master’ first):

$ git branch -d [name\_of\_your\_new\_branch]

Delete the new branch on Github:

$ git push origin :[name\_of\_your\_new\_branch]

Removing commits (ALL commands in ‘master’):

All this does is move the HEAD pointer back however many commits being removed.

To remove the last commit from git, you can simply run git reset --hard HEAD^ If you are removing multiple commits from the top, you can run git reset --hard HEAD~2 to remove the last two commits. You can increase the number to remove even more commits.

If you want to save the commits on a new branch name, then run git branch [newbranchname] before doing the git reset. The new branch would essentially save a copy.

To remove the most recent commit (locally and on Github):

$ git reset --hard HEAD^

$ git push origin -f

Cloning repo & linking w/ Github (SSH):

1. First, check for existing SSH keys:

$ ls -al ~/.ssh

# Lists the files in your .ssh directory, if they exist

Check the directory listing to see if you already have a public SSH key. By default, the filenames of the public keys are one of the following:

* *id\_rsa.pub*
* *id\_ecdsa.pub*
* *id\_ed25519.pub*

If you don't have an existing public and private key pair, or don't wish to use any that are available to connect to GitHub, then generate a new SSH key.

1. To generate a new SSH key:

$ ssh-keygen -t rsa -b 4096 -C "*your\_email@example.com*"

This creates a new SSH key, using the provided email as a label.

When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.

> Enter a file in which to save the key (/c/Users/*you*/.ssh/id\_rsa):*[Press enter]*

At the prompt to type a secure passphrase, just press Enter (no need for a passphrase):

> Enter passphrase (empty for no passphrase): *[Press enter]*

> Enter same passphrase again: *[Press enter]*

1. Next step is to add the SSH key to your Github account:

Locate the hidden .ssh folder, open the file (*‘id\_rsa.pub’*, ‘*id\_ecdsa.pub’*, or ‘*id\_ed25519.pub’*) in your favorite text editor, and copy it to your clipboard.

Go to your Github account and access ‘Settings’ > ‘SSH and GPG Keys’ > ‘New SSH key’. Give the key a unique/memorable title and paste in the key, then select ‘Add SSH key’.

Test your SSH connection (via Terminal):

$ ssh -T git@github.com

# Attempts to ssh to GitHub

You may see a warning like this:

> The authenticity of host 'github.com (IP ADDRESS)' can't be established.

> RSA key fingerprint is 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.

> Are you sure you want to continue connecting (yes/no)?

or like this:

> The authenticity of host 'github.com (IP ADDRESS)' can't be established.

> RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.

> Are you sure you want to continue connecting (yes/no)?

Verify that the fingerprint in the message you see matches one of the messages, then type yes and you should see this:

> Hi [Github\_username]! You've successfully authenticated, but GitHub does

> not provide shell access.

1. Next step is to clone the repo from Github (you should have already created a new/empty repo on Github):

Go to the repo on Github, copy the SSH remote url, and type:

$ git clone git@github.com:Github\_username/repo\_name.git

1. Lastly, set your email address for that repo locally so that your commits will show up as your Github user account on Github. GitHub uses the email address set in your local Git configuration to associate commits pushed from the command line with your GitHub account.

Via Terminal:

Change the current working directory to the local repository where you want to configure the email address that you associate with your Git commits.

Set an email address in Git. Use the email address associated with your Github account.

$ git config user.email "[your\_Github\_account\_email]"

Confirm that you have set the email address correctly in Git:

$ git config user.email

your\_Github\_account\_email