

# git

by Mark Walsh (AIICT)

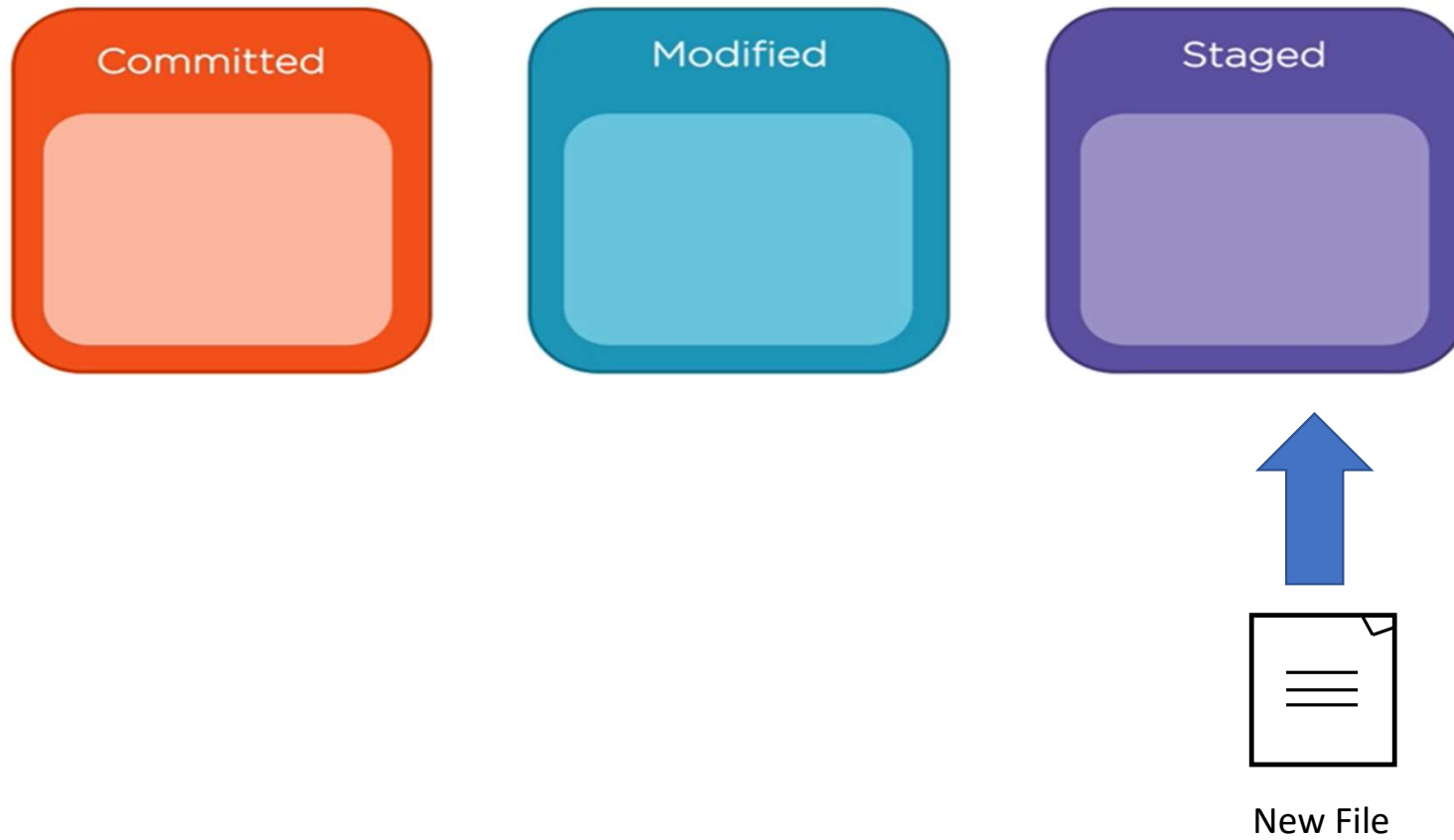
## What is a Version Control System?

- Version Control Systems keep a history of files over time.
- You can see everything that has happened to your repository of files,
  - When files were added.
  - When files were changed.
  - When files were deleted.
  - Who added/changed/deleted the files.
- You can recover a version of the files at a point in time.

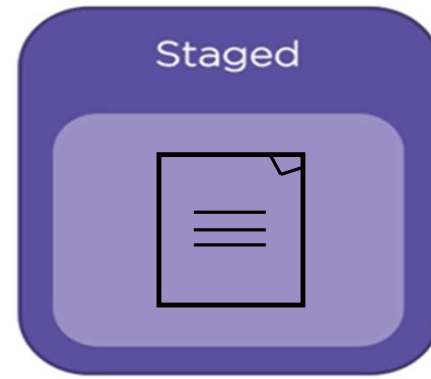
## What is GIT??

- Git is a version control system
- Git is Open Source Software

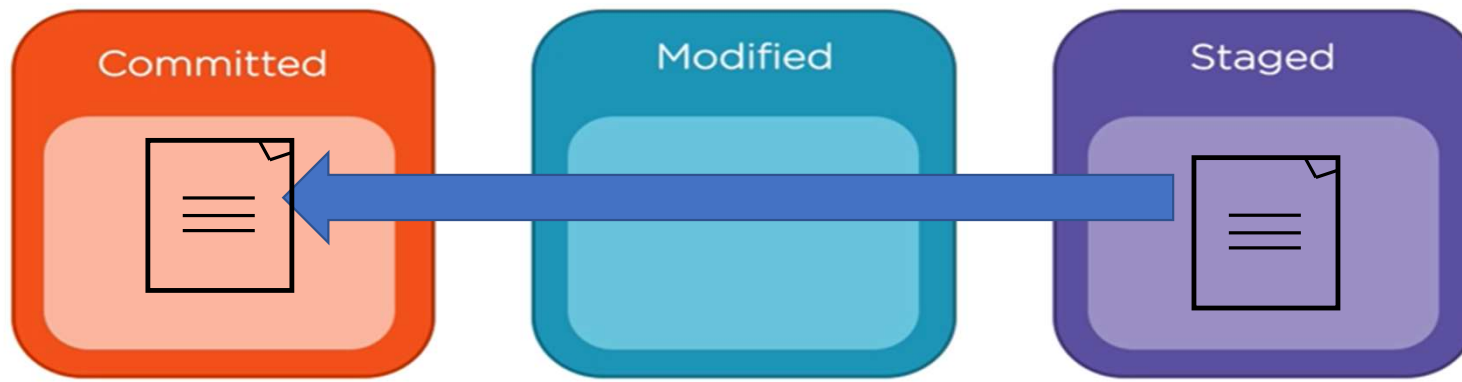
## The Basics of GIT (staging)



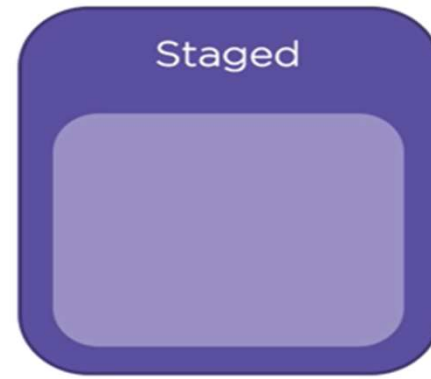
## The Basics of GIT (Staging)



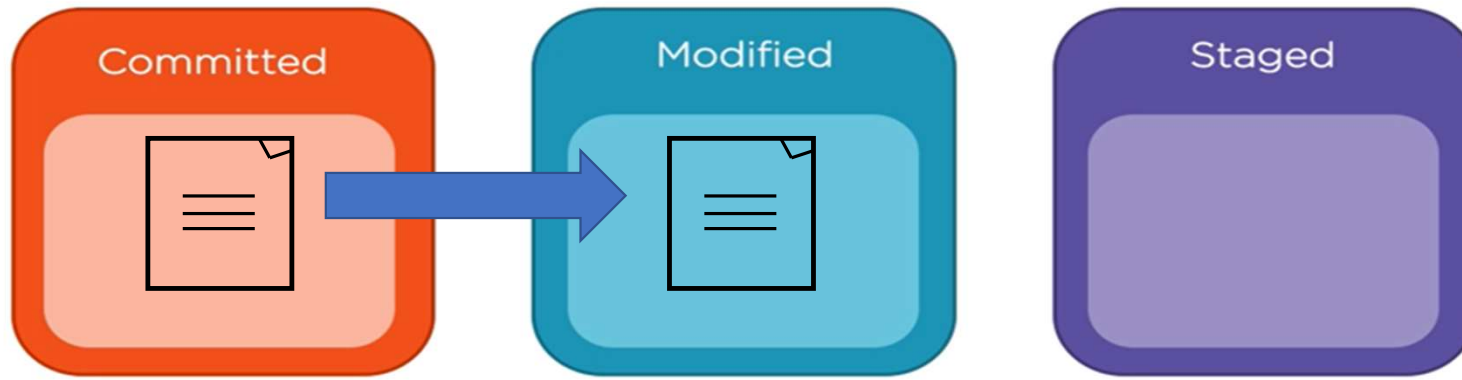
## The Basics of GIT (Commit)



## The Basics of GIT (Commit)

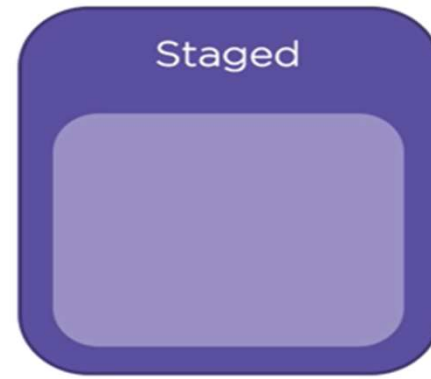
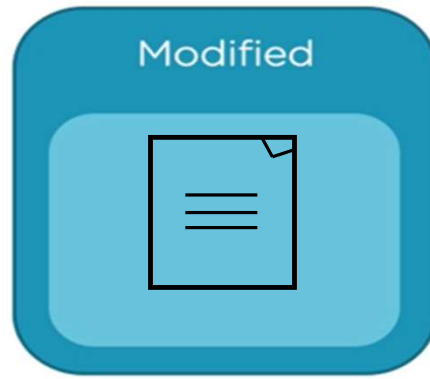


## The Basics of GIT (Modifying)

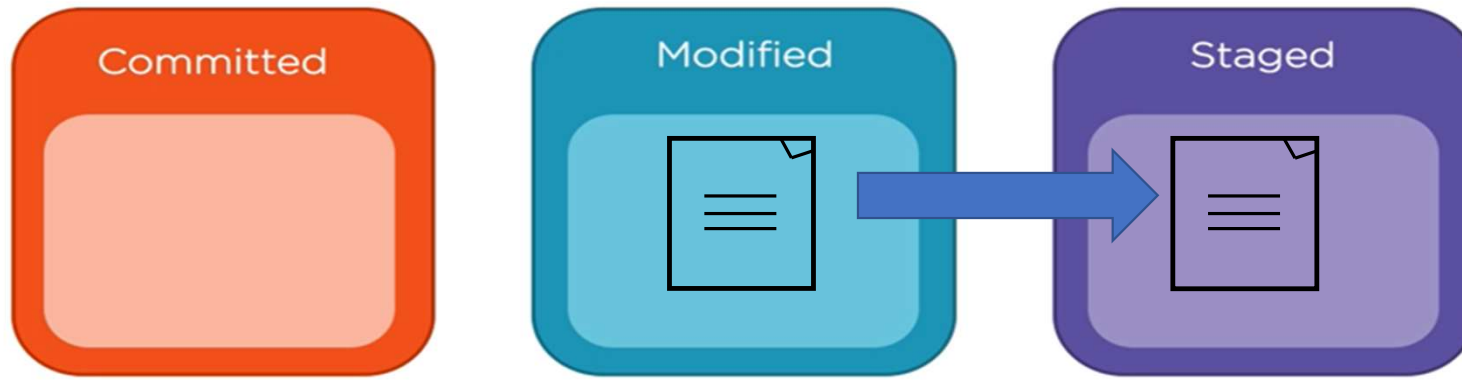




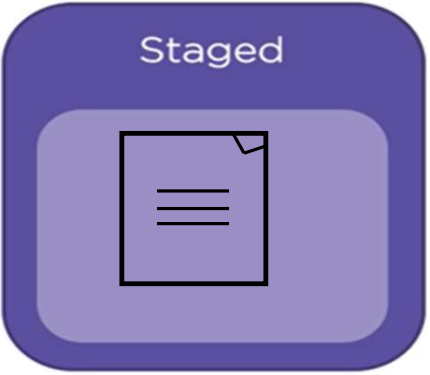
## The Basics of GIT (Modifying)



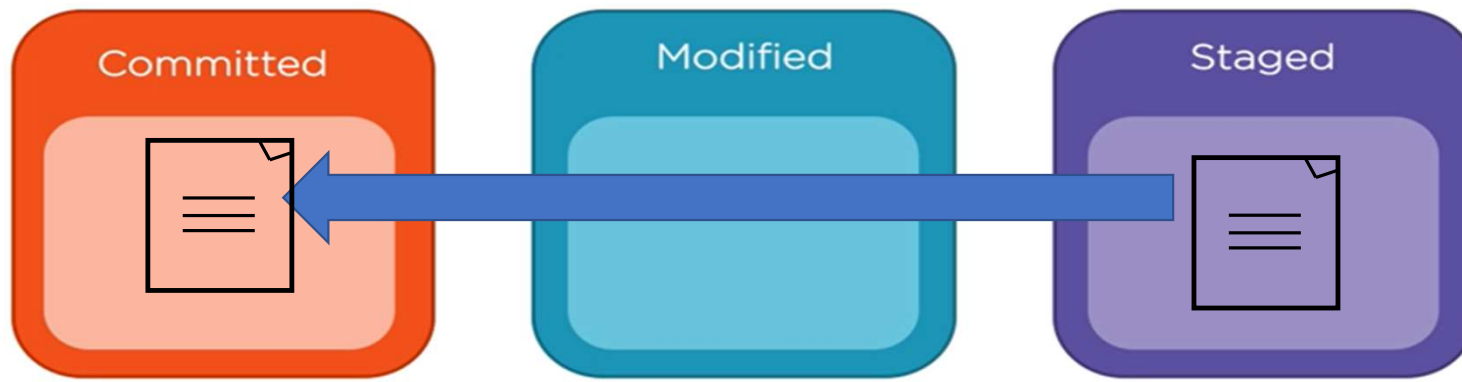
## The Basics of GIT (Modifying)



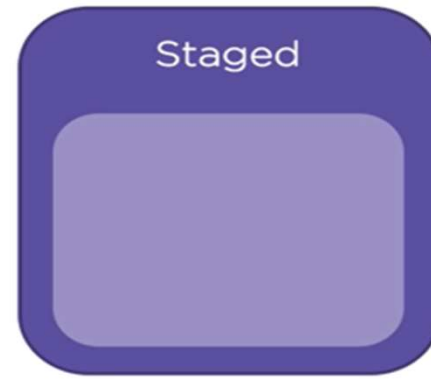
## The Basics of GIT (Modifying)



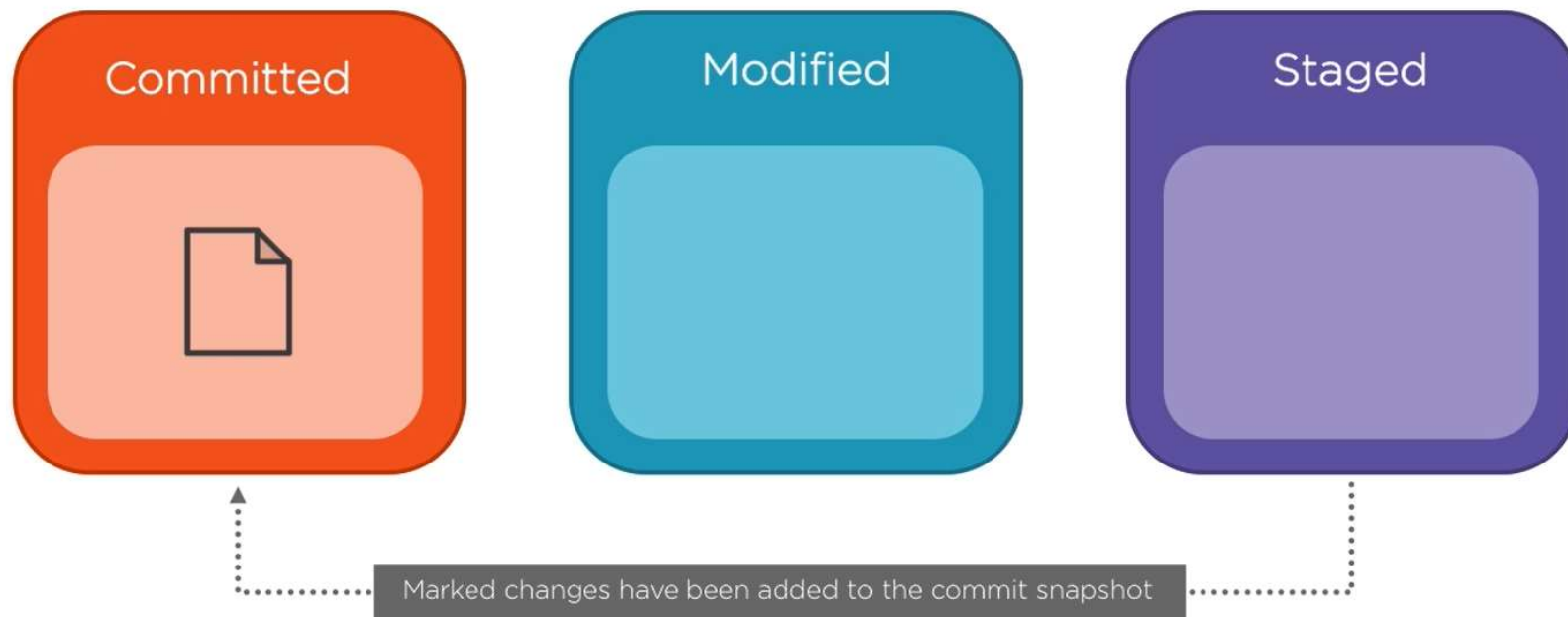
## The Basics of GIT (Commit)



## The Basics of GIT (Commit)



## The Three Stages of a File



Install Git on Windows

<https://git-scm.com/download/win>

# Follow me

- Do what I am doing on the LAB01 virtual machine...



After Install, do some basic config

```
Command Prompt

C:\>git --version
git version 2.32.0.windows.2

C:\>git config --global user.name "Saskia Walsh"

C:\>git config --global user.email "Saskia@Thinkers.com.au"

C:\>git config user.name
Saskia Walsh
```

## Setup your first Git repository

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

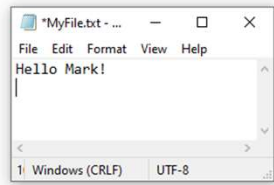
PS C:\Junk\MyProject> git init
Initialized empty Git repository in C:/Junk/MyProject/.git/
PS C:\Junk\MyProject> |
```

That's the repo



<input type="checkbox"/> Name	Date modified	Type
.git	4/07/2022 9:06 AM	File folder

## Adding files, staging



1. add a file

```
PS C:\Junk\MyProject> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        MyFile.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Junk\MyProject> git add MyFile.txt
PS C:\Junk\MyProject> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   MyFile.txt
PS C:\Junk\MyProject> |
```

2. See the file is not tracked by Git Yet

3. Get git to track the file

4. See the file is staged and ready to be committed

## Adding files, committing

```
PS C:\Junk\MyProject> git diff --cached
diff --git a/MyFile.txt b/MyFile.txt
new file mode 100644
index 00000000..d0966f7
--- /dev/null
+++ b/MyFile.txt
@@ -0,0 +1 @@
+Hello Mark!
\ No newline at end of file
```

1. See what is different  
between staged changes  
previous commit

```
PS C:\Junk\MyProject> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   MyFile.txt
```

2. See what will be  
committed

```
PS C:\Junk\MyProject> git commit -m "add first file"
[master (root-commit) 4feeacd] add first file
1 file changed, 1 insertion(+)
create mode 100644 MyFile.txt
PS C:\Junk\MyProject> git status
On branch master
nothing to commit, working tree clean
```

3. Commit changes

Adding files, committing

```
PS C:\Junk\MyProject> git log
commit 4feeacdbdfd364ba9dbcf98b4108fc92256d3e3f (HEAD -> master)
Author: Mark Walsh <Mark.Walsh@ddls.com.au>
Date: Mon Jul 4 09:23:00 2022 +1000

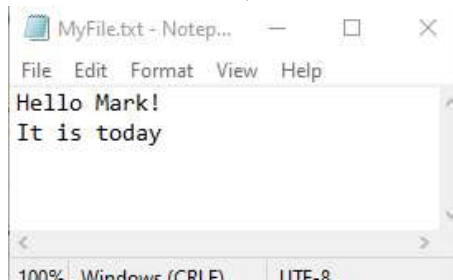
    add first file
```

1. See History

## Changing the files in a repo

```
PS C:\Junk\MyProject> git status
On branch master
nothing to commit, working tree clean
```

Before modification



Modify the file

```
PS C:\Junk\MyProject> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   MyFile.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

See that git knows it's modified

## Changing the files in a repo

```
PS C:\Junk\MyProject> git diff HEAD
diff --git a/MyFile.txt b/MyFile.txt
index d0966f7..ecfdcaa 100644
--- a/MyFile.txt
+++ b/MyFile.txt
@@ -1,2 @@
-Hello Mark!
\ No newline at end of file
+Hello Mark!^M
+It is Today
\ No newline at end of file
```

Display what is different  
between unstaged changes  
and last commit

```
PS C:\Junk\MyProject> git add .
PS C:\Junk\MyProject> git diff --cached
diff --git a/MyFile.txt b/MyFile.txt
index d0966f7..ecfdcaa 100644
--- a/MyFile.txt
+++ b/MyFile.txt
@@ -1,2 @@
-Hello Mark!
\ No newline at end of file
+Hello Mark!^M
+It is Today
\ No newline at end of file
```

Stage changes

Display what is different  
between staged changes  
and last commit

```
PS C:\Junk\MyProject> git commit -m "Add additional line to file"
[master 3e10de2] Add additional line to file
1 file changed, 2 insertions(+), 1 deletion(-)
```

Commit changes to the repo  
(then run git log)

Displaying differences before doing a commit

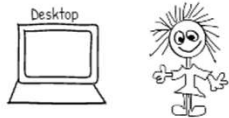
Git diff --Staged



also: `git diff --staged --no-renames`



## Summary of git basics



1. 

```
PS C:\Junk\MyProject> git init
```

  
Initialized empty Git repository in C:/Junk/MyProject/.git/ **Initialise a repository**
2.  **Add a file**  
MyFile.txt
3. 

```
PS C:\Junk\MyProject> git add .
```

**Stage to Commit**
4. 

```
PS C:\Junk\MyProject> git commit -m "add myfile"
```

  
[master (root-commit) 6ed6b48] add myfile  
1 file changed, 1 insertion(+)  
create mode 100644 MyFile.txt **Commit**
5.  **Change the file**  
MyFile.txt
6. 

```
PS C:\Junk\MyProject> git add .
```

**Stage to Commit**
7. 

```
PS C:\Junk\MyProject> git commit -m "change myfile"
```

  
[master e72570c] change myfile  
1 file changed, 1 insertion(+), 1 deletion(-) **Commit**

### 8. Show a history with details

```
PS C:\Junk\MyProject> git log -p
```

commit e72570cca8e3f8836f7f2f3c9a7a70377829d571 (HEAD -> master)  
Author: Saskia Walsh <Saskia@Thinkers.com.au>  
Date: Fri Jul 1 13:27:40 2022 +1000

change myfile

diff --git a/MyFile.txt b/MyFile.txt  
index 05a682b..d0966f7 100644  
index 05a682b..d0966f7 100644  
--- a/MyFile.txt  
+++ b/MyFile.txt  
@@ -1,1 @@  
-Hello!  
 \ No newline at end of file  
+Hello Mark!  
 \ No newline at end of file

commit 6ed6b48110151e77e601450fa93a3f53ed0309f2  
Author: Saskia Walsh <Saskia@Thinkers.com.au>  
Date: Fri Jul 1 13:22:51 2022 +1000

add myfile

diff --git a/MyFile.txt b/MyFile.txt  
new file mode 100644  
index 0000000..05a682b  
--- /dev/null  
+++ b/MyFile.txt  
@@ -0,0 +1 @@  
+Hello!  
 \ No newline at end of file

initial commit

commit after change

# Follow me

OK,

1. now add another file
2. get git to track the file (*git add*)
3. Commit your new file (*git commit*)
4. Make a change to your new file
5. Show the differences (*git diff HEAD*)
6. Stage the changes (*git add*)
7. Commit the changes.

## Git Key Concepts, Commit Hashes

PS C:\Junk\MyProject> **git log -p**

**commit** e72570cca8e3f8836f7f2f3c9a7a70377829d571 (HEAD -> master)  
Author: Saskia Walsh <Saskia@Thinkers.com.au>  
Date: Fri Jul 1 13:27:40 2022 +1000

change myfile

diff --git a/MyFile.txt b/MyFile.txt  
index 05a682b..d0966f7 100644  
index 05a682b..d0966f7 100644  
--- a/MyFile.txt  
+++ b/MyFile.txt  
@@ -1,1 @@  
-Hello!  
\ No newline at end of file  
+Hello Mark!  
\ No newline at end of file

**commit** 6ed6b48110151e77e601450fa93a3f53ed0309f2  
Author: Saskia Walsh <Saskia@Thinkers.com.au>  
Date: Fri Jul 1 13:22:51 2022 +1000

add myfile

diff --git a/MyFile.txt b/MyFile.txt  
new file mode 100644  
index 0000000..05a682b  
--- /dev/null  
+++ b/MyFile.txt  
@@ -0,0 +1 @@  
+Hello!  
\ No newline at end of file

commit after change

initial commit

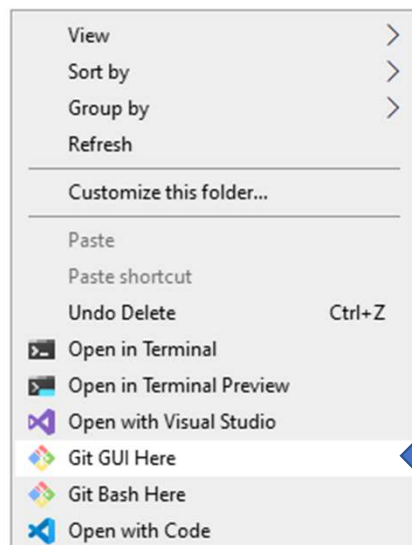
What is that long series of characters?

# Follow me

- Do what I am doing on the LAB01 virtual machine...

How about some kind of graphical tool? GitGui

<input type="checkbox"/> Name	Date modified	Type	Size
.git	4/07/2022 9:35 AM	File folder	
MyFile.txt	4/07/2022 9:28 AM	Text Document	1 KB



How about some kind of graphical tool? GitGui

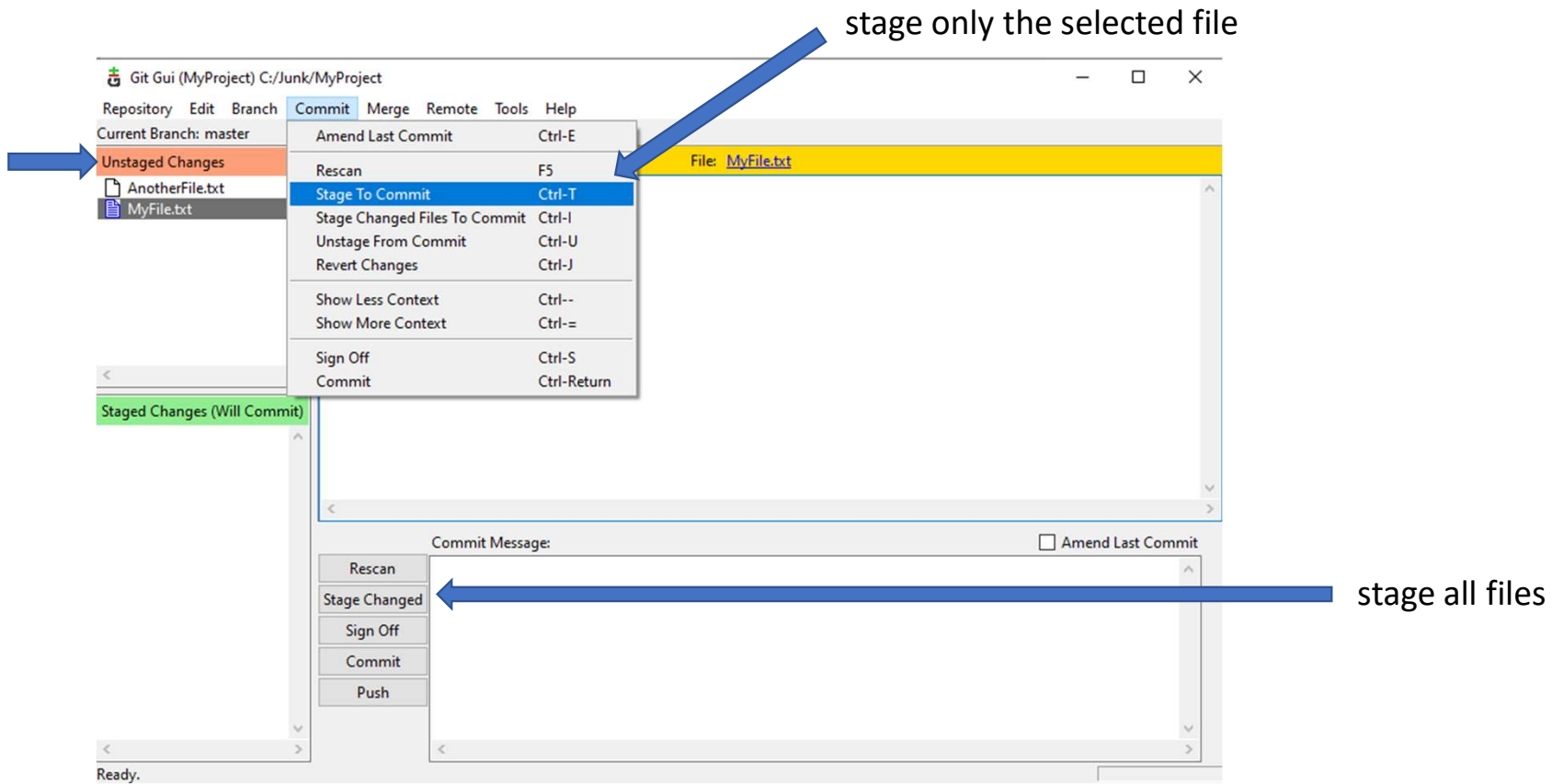
The diagram illustrates the steps to use GitGui, showing a file explorer, two text editors, and a context menu.

**1. Add a file**: A file explorer window shows a directory with a `.git` folder and two files: `AnotherFile.txt` and `MyFile.txt`. A blue arrow points from `AnotherFile.txt` to a text editor window titled `AnotherFile.tx...` containing the text "Yes, it's another file".

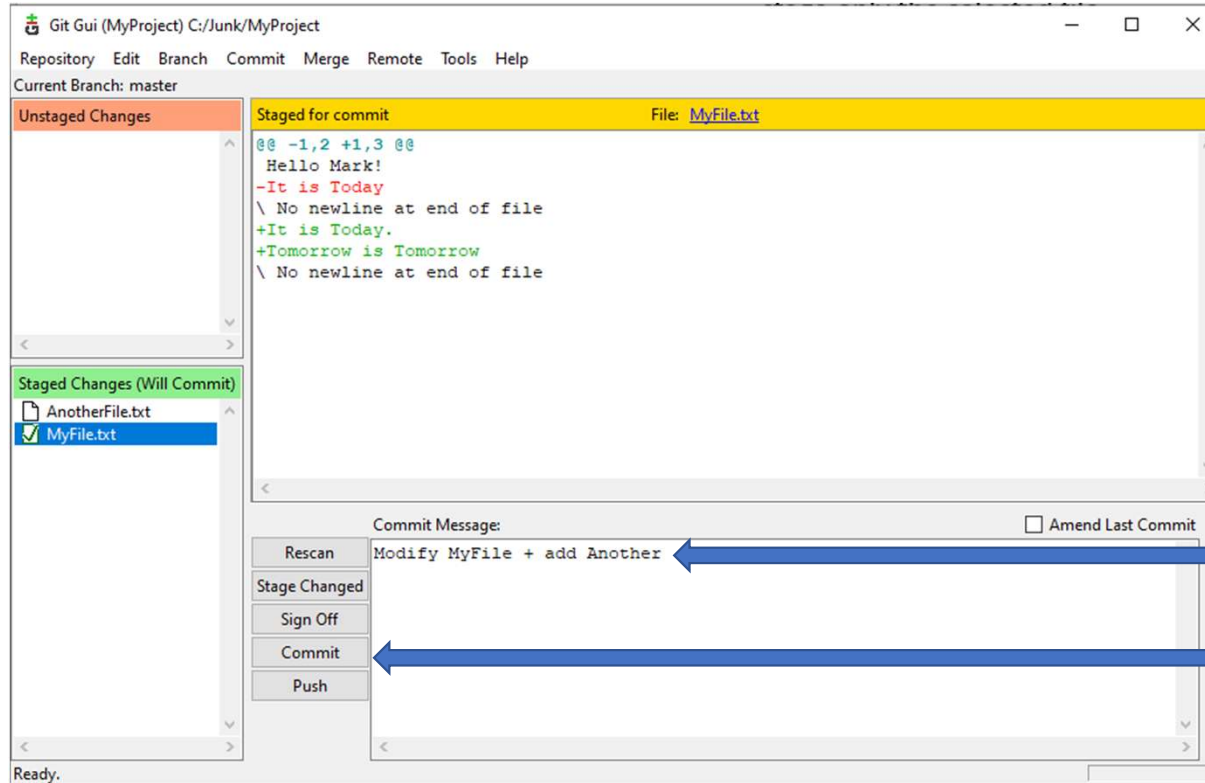
**2. Change a file**: A blue arrow points from `MyFile.txt` in the file explorer to a text editor window titled `MyFile.txt - N...` containing the text "Hello Mark!", "It is Today.", and "Tomorrow is Tomorrow".

**3. Open GitGui**: A blue arrow points from a context menu with two options: "Git GUI Here" and "Git Bash Here".

How about some kind of graphical tool? GitGui



How about some kind of graphical tool? GitGui



The Commit message

Commit all staged files



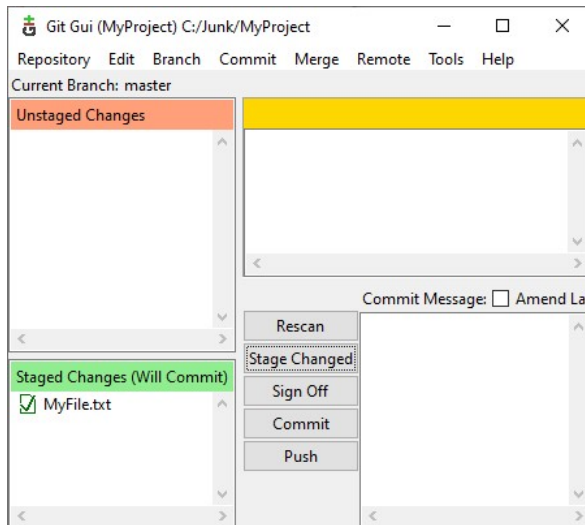
## A useful git command

git checkout .

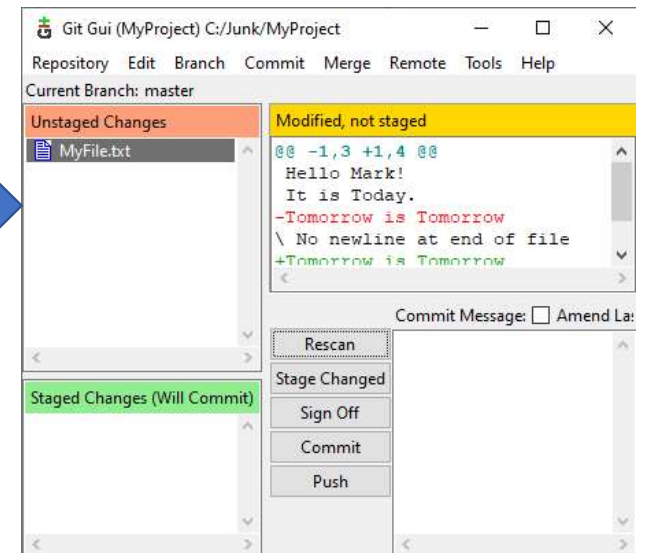
git checkout *filename.suffix*

If you have changed files but not yet run ***git add***, you can undo your local changes with ***git checkout***

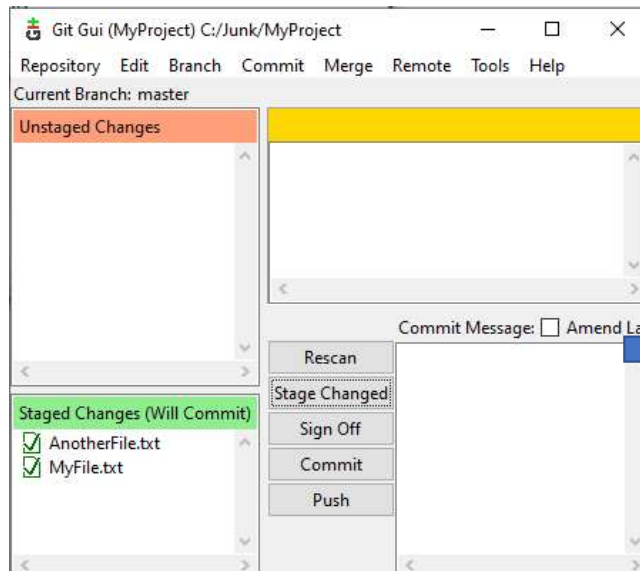
How do I *unstage* all staged files?



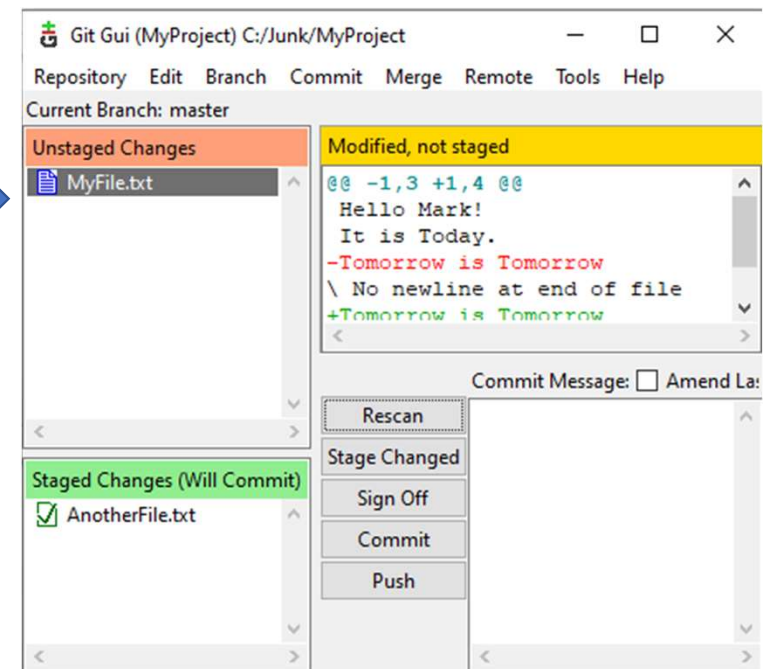
```
PS C:\Junk\MyProject> git reset HEAD
Unstaged changes after reset:
M   MyFile.txt
```



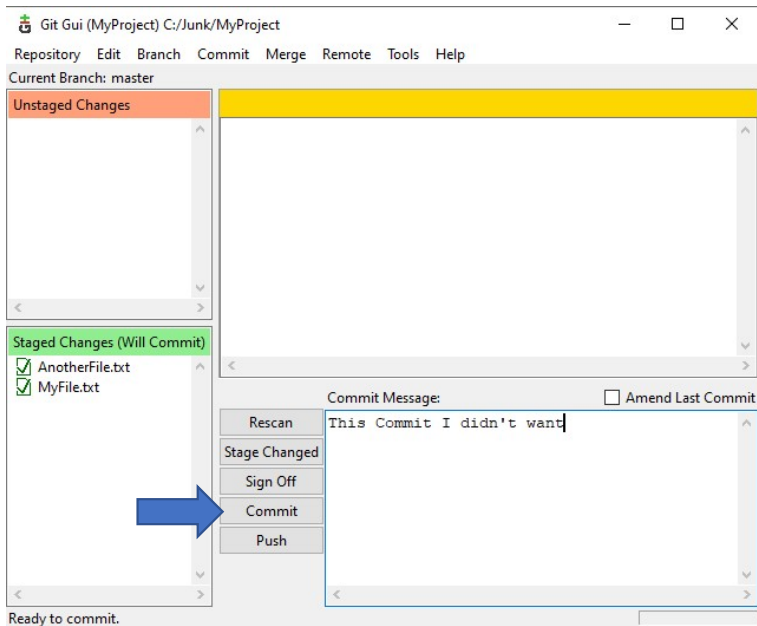
How do I *unstage* one staged file?



```
PS C:\Junk\MyProject> git reset MyFile.txt HEAD
Unstaged changes after reset:
M   MyFile.txt
```



## How do I *undo* a (local) Commit?

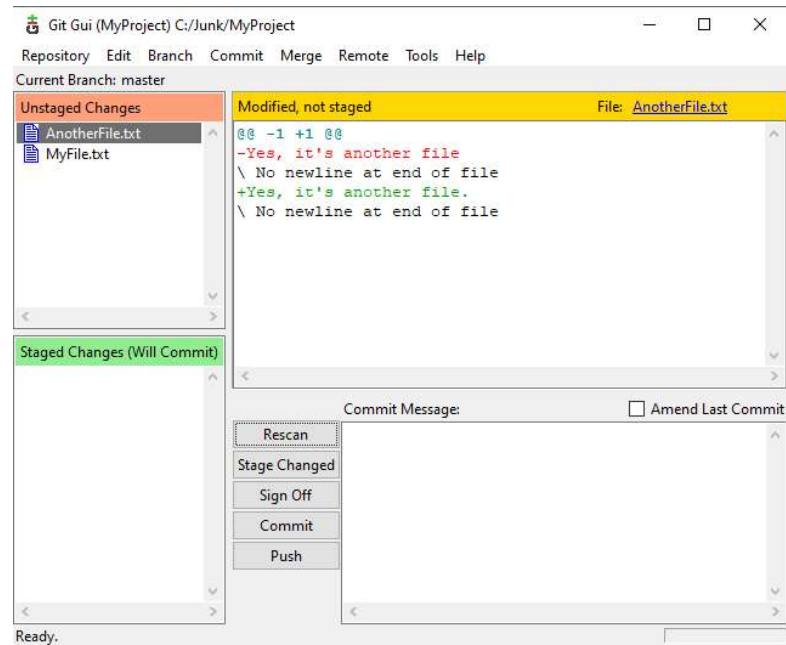


```
PS C:\Junk\MyProject> git log
commit f6212edb29c00734d77fc313fcd23a062ce9efe4 (HEAD -> master)
Author: Mark Walsh <Mark.Walsh@ddls.com.au>
Date: Mon Jul 4 10:31:44 2022 +1000

    This Commit I didn't want

commit 18ee3a6e3d5e94649104d20b29749945b17dabc0
Author: Mark Walsh <Mark.Walsh@ddls.com.au>
Date: Mon Jul 4 10:14:45 2022 +1000

    Modify MyFile + add Another
```



```
PS C:\Junk\MyProject> git reset HEAD~
Unstaged changes after reset:
M    AnotherFile.txt
M    MyFile.txt
```

git reset



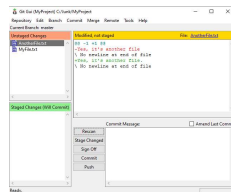
Let's try these while having GitGui open to see what they do



git reset --soft *SHA1*

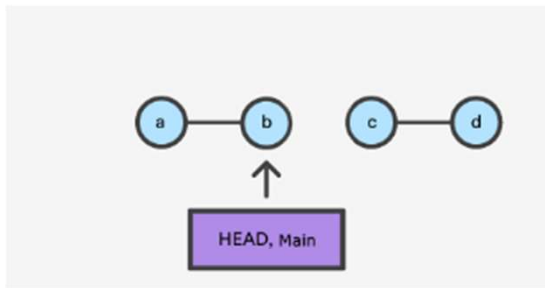
git reset --mixed *SHA1*

git reset --hard *SHA1*

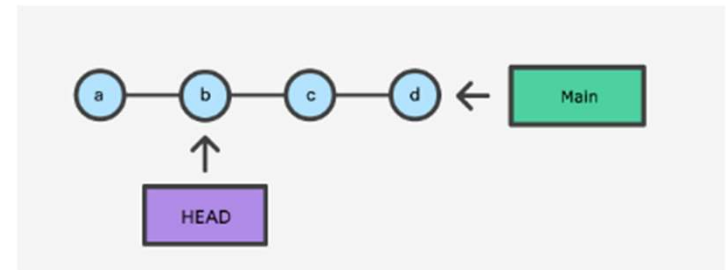


## git reset vs git checkout

git reset b



git checkout b



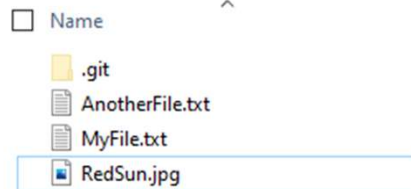
Git reset will never delete a commit, however, commits can become 'orphaned' which means there is no direct path from a ref to access them. **Git will permanently delete any orphaned commits after it runs the internal garbage collector**

# Do that again?

1. Make a change to a file
2. Undo the changes (*git checkout .*)
3. Make a change to a file
4. Stage the changed file (*git add*)
5. Un-stage the changed file (*git reset HEAD*)
6. Make a change to a file
7. Stage the changed file (*git add*)
8. Commit the changes (*git commit*)
9. Undo the commit (*git reset HEAD~*)

Can I put binary files in my repo?

Yes you can, but they are not *diffable*. They can also make your repo large.



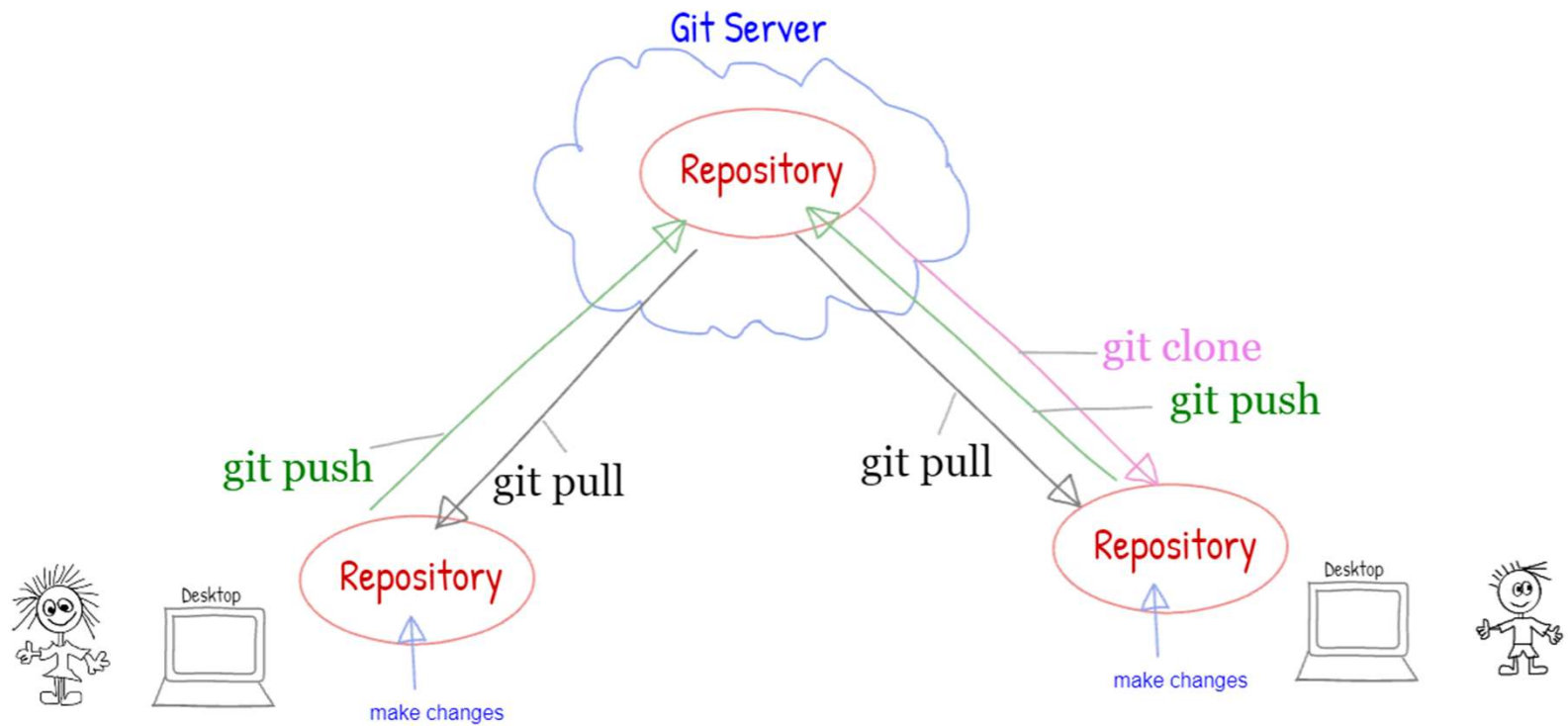
```
PS C:\Junk\MyProject> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        RedSun.jpg

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Junk\MyProject> git add RedSun.jpg
PS C:\Junk\MyProject> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   RedSun.jpg

PS C:\Junk\MyProject> git commit -m "Add Red Sun"
[master 64d7ba6] Add Red Sun
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 RedSun.jpg
```



## Cloning repo's and collaborating



## Cloning repo's and collaborating



<https://github.com/>

**GitHub** is not the same thing as **Git**:

- GitHub is a fully managed Service owned by Microsoft.
- Github uses Git as its Version Control System.
- Github is a place where you can put your *origin* repo's.
- Github helps team members collaborate, including,
  - PR's (pull requests).
  - *Github Flow* for managing changes.
  - *Actions* for CI/CD pipelines.

# Follow me (using Lab01 VM)

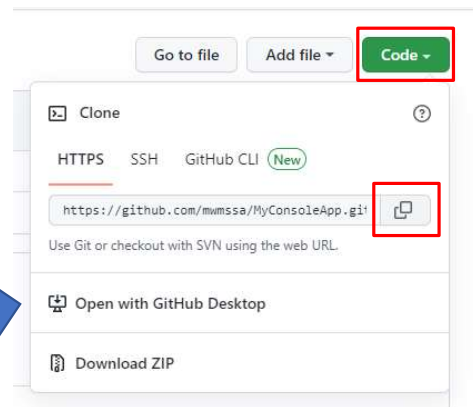
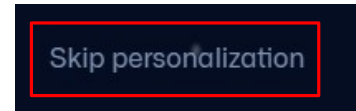
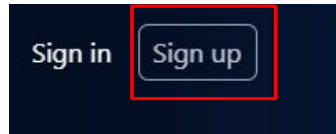
## Overview:

1. Create a new repo in Github (include gitignore and readme)
2. Git Clone
3. Add your application code
4. Stage files
5. Commit files
6. git push

## Cloning repo's and collaborating



<https://github.com/>



```
PS C:\Junk> git clone https://github.com/mwmssa/MyConsoleApp.git
Cloning into 'MyConsoleApp'...
info: please complete authentication in your browser...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

□ Name ^

- .git
- .gitignore
- README.md

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \* [mwmssa](#) / Repository name \* [MyConsoleApp](#) ✓

Great repository names are short and memorable. Need inspiration? How about [animated-succotash](#)?

Description (optional)

☐ Public  
Anyone on the internet can see this repository. You choose who can commit.

☒ Private  
You choose who can see and commit to this repository.

Initialize this repository with:  
Skip this step if you're importing an existing repository.

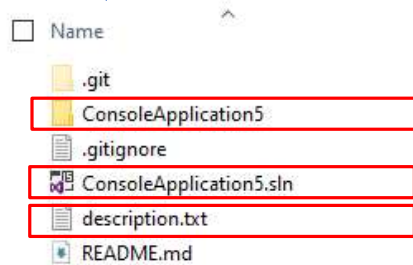
☒ Add a README file  
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore  
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: [VisualStudio](#)

Create repository

Create Application -> Stage -> Commit -> Push

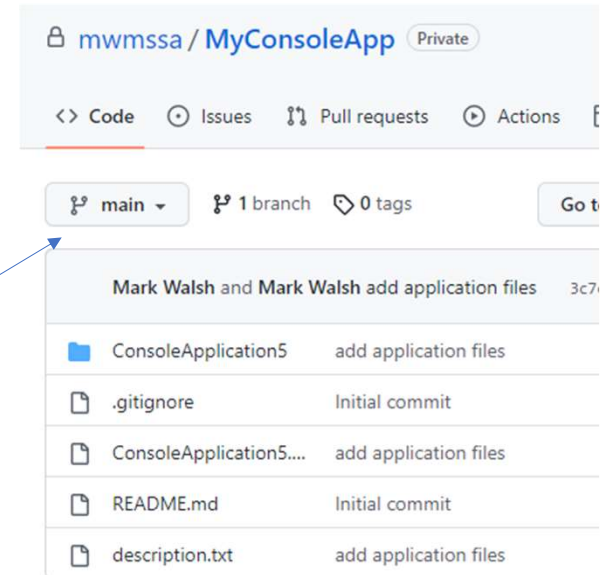


I added these

```
PS C:\Junk\MyConsoleApp> git add .
PS C:\Junk\MyConsoleApp> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   ConsoleApplication5.sln
        new file:   ConsoleApplication5/App.config
        new file:   ConsoleApplication5/ConsoleApplication5.csproj
        new file:   ConsoleApplication5/Program.cs
        new file:   ConsoleApplication5/Properties/AssemblyInfo.cs
        new file:   description.txt

PS C:\Junk\MyConsoleApp> git commit -m "add application files"
[main 3c7e374] add application files
 6 files changed, 173 insertions(+)
 create mode 100644 ConsoleApplication5.sln
 create mode 100644 ConsoleApplication5/App.config
 create mode 100644 ConsoleApplication5/ConsoleApplication5.csproj
 create mode 100644 ConsoleApplication5/Program.cs
 create mode 100644 ConsoleApplication5/Properties/AssemblyInfo.cs
 create mode 100644 description.txt
PS C:\Junk\MyConsoleApp> git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (10/10), 3.23 KiB | 661.00 KiB/s, done.
Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/mwmssa/MyConsoleApp.git
bf9caf6..3c7e374  main -> main
```



How can I copy this repo to another device?

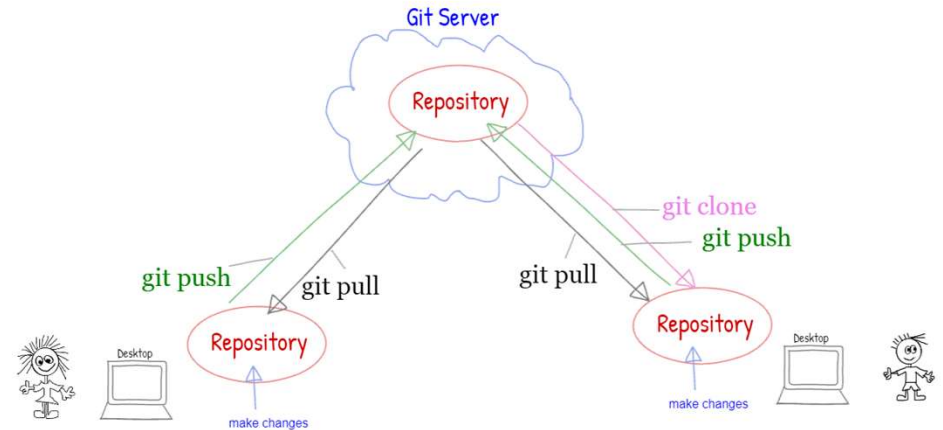
# Do it again (using Lab01)

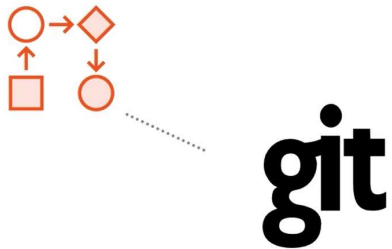
## Overview:

1. Create a new **public** repo in Github (include gitignore and readme)
2. Git Clone
3. Add your application code
4. Stage files
5. Commit files
6. git push

## OK, Let's Collaborate

1. Please add your github user name to the Microsoft Teams chat.
2. I will invite you to an origin repo.
3. Find the invitation in your email and accept it.
4. Git clone the repo.
5. I will assign a task to each of you.
6. Change the application code in your local repo and test it.
7. git stage -> commit -> push
8. Tell me when you're done.

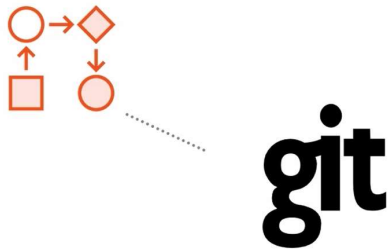




More on Git Later. For now,

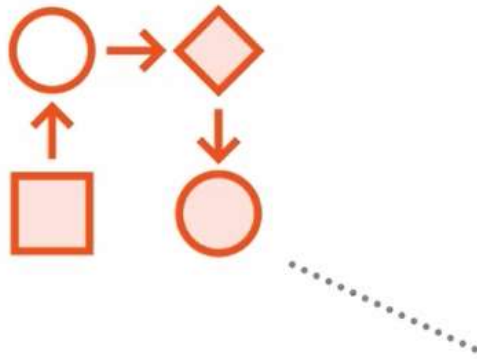
- You have a place to store your work even if your lab ends.
- Do this:
  1. Create a new *Public* repo in Github.
  2. Save all





## Ad-hoc exercises

- Do this:
  1. Create a new *Public* repo in Github.
  2. Save all your work in your public repo.
  3. We can review each other's work and help each other.

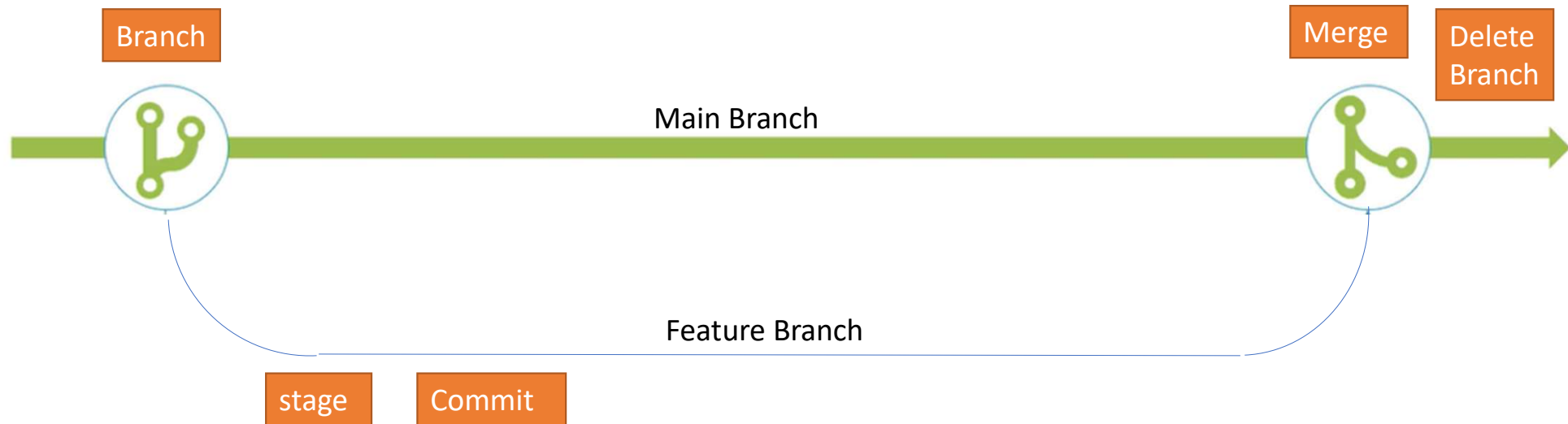


MORE

**git**

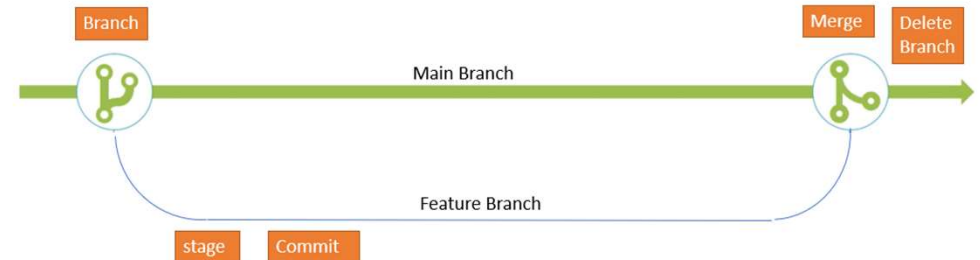
by Mark Walsh (AIICT)

OK, Time to learn how to do things better using branches



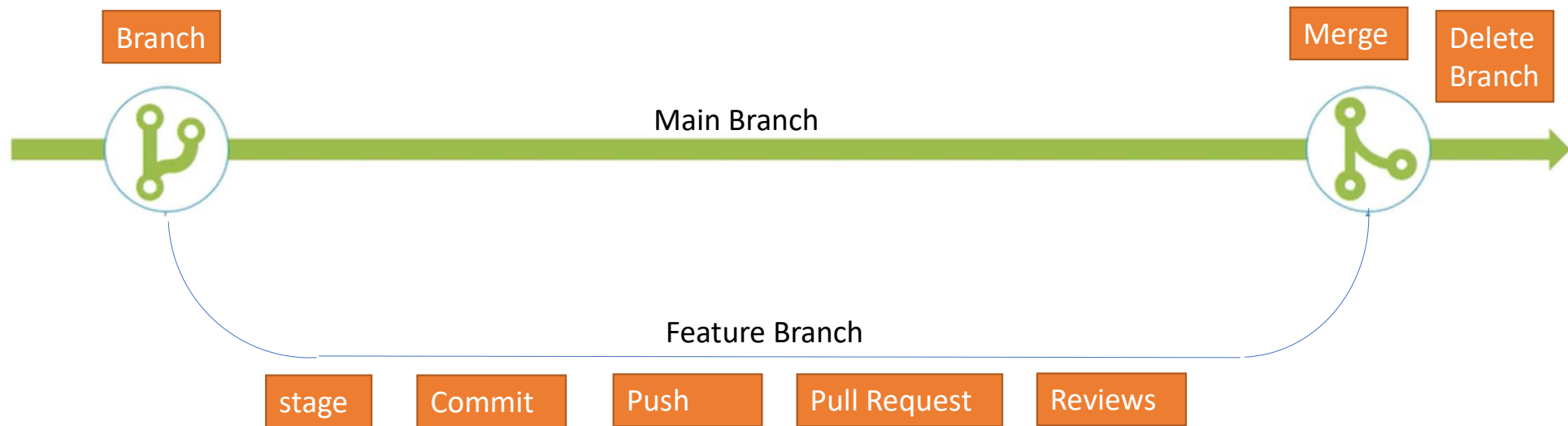
## Branching on your local repo

1. Create a local Repo
2. Add some files.
3. git stage -> commit



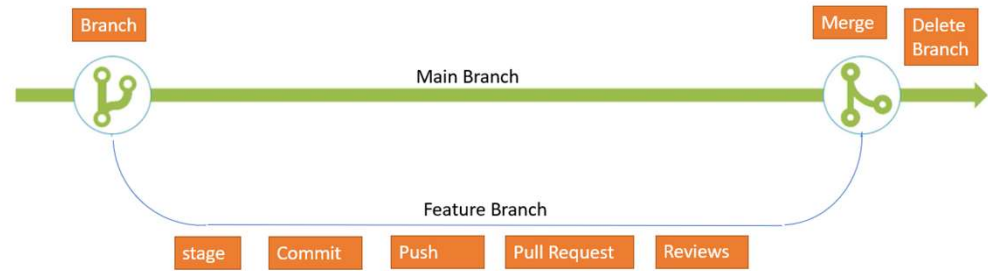
4. Create a feature branch (git branch feature-111)
5. Checkout the feature branch (git checkout feature-111)
6. Ensure you're on the feature branch (git status)
7. Make some changes to the files.
8. git stage -> commit
9. Make some changes to the files.
10. git stage -> commit
11. list branches (git branch)
12. Switch to master branch (git checkout master)
13. Merge feature into master (git merge feature-111)
14. Delete the feature branch (git branch --delete feature-111)

OK, Time to learn how to do that better using branches and GitHub Flow



## OK, Let's Collaborate

1. Git clone my repo.
2. I will create a feature branch for each of you.
3. run **git branch -r** to see what branches there are on git hub
4. Run git branch to see what branches you have. You can run **git fetch origin *branchname*** to get the branch into your local repo if it's not there.
5. Checkout your feature branch.
6. Change the application code in your local repo and test it.
7. git stage -> commit on the feature branch
8. Push the feature branch to the origin
9. Raise a PR



## Summary of key Git Commands

### GIT BASICS

<code>git init</code> <code>&lt;directory&gt;</code>	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
<code>git clone &lt;repo&gt;</code>	Clone repo located at <code>&lt;repo&gt;</code> onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
<code>git config</code> <code>user.name &lt;name&gt;</code>	Define author name to be used for all commits in current repo. Devs commonly use <code>--global</code> flag to set config options for current user.
<code>git add</code> <code>&lt;directory&gt;</code>	Stage all changes in <code>&lt;directory&gt;</code> for the next commit. Replace <code>&lt;directory&gt;</code> with a <code>&lt;file&gt;</code> to change a specific file.
<code>git commit -m</code> " <code>&lt;message&gt;</code> "	Commit the staged snapshot, but instead of launching a text editor, use <code>&lt;message&gt;</code> as the commit message.
<code>git status</code>	List which files are staged, unstaged, and untracked.
<code>git log</code>	Display the entire commit history using the default format. For customization see additional options.
<code>git diff</code>	Show unstaged changes between your index and working directory.

### UNDOING CHANGES

<code>git revert</code> <code>&lt;commit&gt;</code>	Create new commit that undoes all of the changes made in <code>&lt;commit&gt;</code> , then apply it to the current branch.
<code>git reset &lt;file&gt;</code>	Remove <code>&lt;file&gt;</code> from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
<code>git clean -n</code>	Shows which files would be removed from working directory. Use the <code>-f</code> flag in place of the <code>-n</code> flag to execute the clean.

### REWRITING GIT HISTORY

<code>git commit</code> <code>--amend</code>	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
<code>git rebase &lt;base&gt;</code>	Rebase the current branch onto <code>&lt;base&gt;</code> . <code>&lt;base&gt;</code> can be a commit ID, branch name, a tag, or a relative reference to HEAD.
<code>git reflog</code>	Show a log of changes to the local repository's HEAD. Add <code>--relative-date</code> flag to show date info or <code>--all</code> to show all refs.

### GIT BRANCHES

<code>git branch</code>	List all of the branches in your repo. Add a <code>&lt;branch&gt;</code> argument to create a new branch with the name <code>&lt;branch&gt;</code> .
<code>git checkout -b</code> <code>&lt;branch&gt;</code>	Create and check out a new branch named <code>&lt;branch&gt;</code> . Drop the <code>-b</code> flag to checkout an existing branch.
<code>git merge &lt;branch&gt;</code>	Merge <code>&lt;branch&gt;</code> into the current branch.

### REMOTE REPOSITORIES

<code>git remote add</code> <code>&lt;name&gt; &lt;url&gt;</code>	Create a new connection to a remote repo. After adding a remote, you can use <code>&lt;name&gt;</code> as a shortcut for <code>&lt;url&gt;</code> in other commands.
<code>git fetch</code> <code>&lt;remote&gt; &lt;branch&gt;</code>	Fetches a specific <code>&lt;branch&gt;</code> , from the repo. Leave off <code>&lt;branch&gt;</code> to fetch all remote refs.
<code>git pull &lt;remote&gt;</code>	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
<code>git push</code> <code>&lt;remote&gt; &lt;branch&gt;</code>	Push the branch to <code>&lt;remote&gt;</code> , along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

## Summary of key Git Commands

### GIT CONFIG

<code>git config --global user.name &lt;name&gt;</code>	Define the author name to be used for all commits by the current user.
<code>git config --global user.email &lt;email&gt;</code>	Define the author email to be used for all commits by the current user.
<code>git config --global alias. &lt;alias-name&gt; &lt;git-command&gt;</code>	Create shortcut for a Git command. E.g. <code>alias.glog "log --graph --oneline"</code> will set "git glog" equivalent to "git log --graph --oneline".
<code>git config --system core.editor &lt;editor&gt;</code>	Set text editor used by commands for all users on the machine. <editor> arg should be the command that launches the desired editor (e.g., vi).
<code>git config --global --edit</code>	Open the global configuration file in a text editor for manual editing.

### GIT LOG

<code>git log --&lt;limit&gt;</code>	Limit number of commits by <limit>. E.g. "git log -5" will limit to 5 commits.
<code>git log --oneline</code>	Condense each commit to a single line.
<code>git log -p</code>	Display the full diff of each commit.
<code>git log --stat</code>	Include which files were altered and the relative number of lines that were added or deleted from each of them.
<code>git log --author="&lt;pattern&gt;"</code>	Search for commits by a particular author.
<code>git log --grep="&lt;pattern&gt;"</code>	Search for commits with a commit message that matches <pattern>.
<code>git log &lt;since&gt;..&lt;until&gt;</code>	Show commits that occur between <since> and <until>. Args can be a commit ID, branch name, HEAD, or any other kind of revision reference.
<code>git log -- &lt;file&gt;</code>	Only display commits that have the specified file.
<code>git log --graph --decorate</code>	--graph flag draws a text based graph of commits on left side of commit msgs. --decorate adds names of branches or tags of commits shown.

### GIT DIFF

<code>git diff HEAD</code>	Show difference between working directory and last commit.
<code>git diff --cached</code>	Show difference between staged changes and last commit

### GIT RESET

<code>git reset</code>	Reset staging area to match most recent commit, but leave the working directory unchanged.
<code>git reset --hard</code>	Reset staging area and working directory to match most recent commit and <b>overwrites all changes</b> in the working directory.
<code>git reset &lt;commit&gt;</code>	Move the current branch tip backward to <commit>, reset the staging area to match, but leave the working directory alone.
<code>git reset --hard &lt;commit&gt;</code>	Same as previous, but resets both the staging area & working directory to match. <b>Deletes</b> uncommitted changes, and <b>all commits after</b> <commit>.

### GIT REBASE

<code>git rebase -i &lt;base&gt;</code>	Interactively rebase current branch onto <base>. Launches editor to enter commands for how each commit will be transferred to the new base.
---	---

### GIT PULL

<code>git pull --rebase &lt;remote&gt;</code>	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.
---	---

### GIT PUSH

<code>git push &lt;remote&gt; --force</code>	Forces the git push even if it results in a non-fast-forward merge. Do not use the --force flag unless you're absolutely sure you know what you're doing.
<code>git push &lt;remote&gt; --all</code>	Push all of your local branches to the specified remote.
<code>git push &lt;remote&gt; --tags</code>	Tags aren't automatically pushed when you push a branch or use the --all flag. The --tags flag sends all of your local tags to the remote repo.

Source: [https://wac-cdn.atlassian.com/dam/jcr:e7e22f25-bba2-4ef1-a197-53f46b6df4a5/SWTM-2088\\_Atlassian-Git-Cheatsheet.pdf?cdnVersion=418](https://wac-cdn.atlassian.com/dam/jcr:e7e22f25-bba2-4ef1-a197-53f46b6df4a5/SWTM-2088_Atlassian-Git-Cheatsheet.pdf?cdnVersion=418)



Ok, that's all for now

You know enough to be dangerous

If ever you create anything on the lab VM  
be sure to commit it to a GitHub repo before  
closing the lab!!!