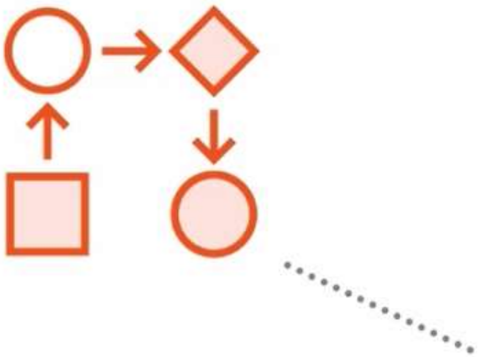


git

by Mark Walsh (AIICT)

What is a Version Control System?

- Version Control Systems keep a history of files over time.

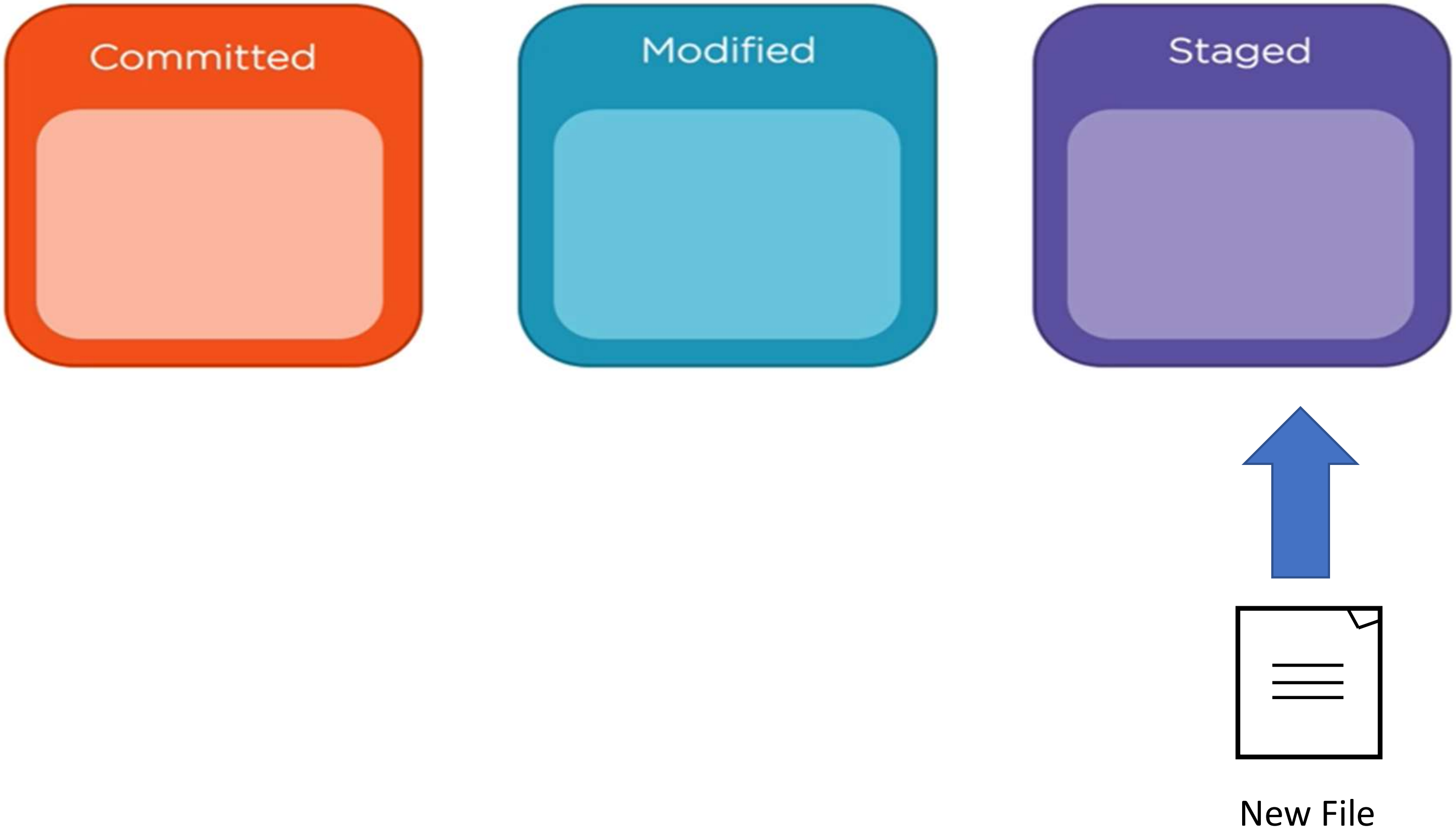
- You can see everything that has happened to your repository of files,
  - When files were added.
  - When files were changed.
  - When files were deleted.
  - Who added/changed/deleted the files.

- You can recover a version of the files at a point in time.

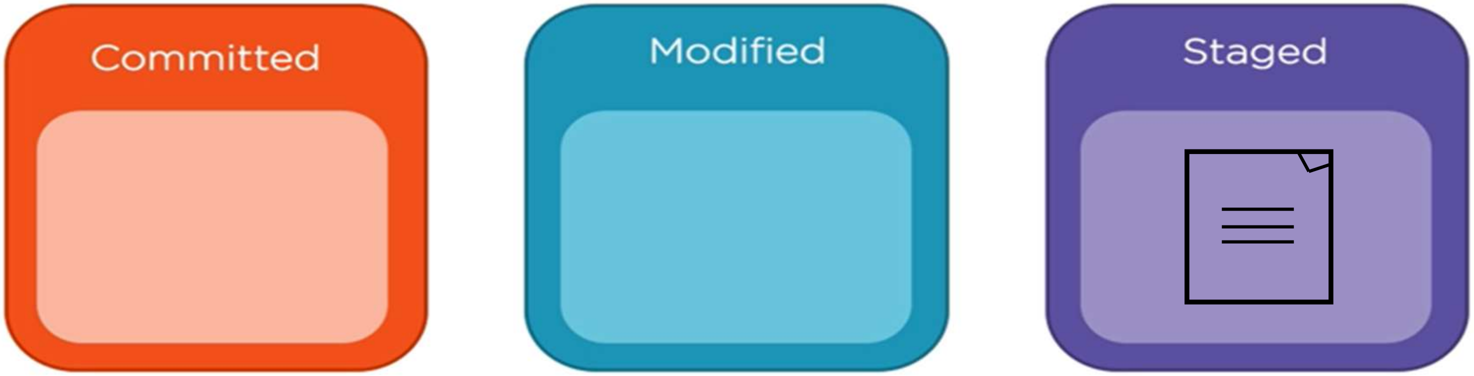
What is GIT??

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

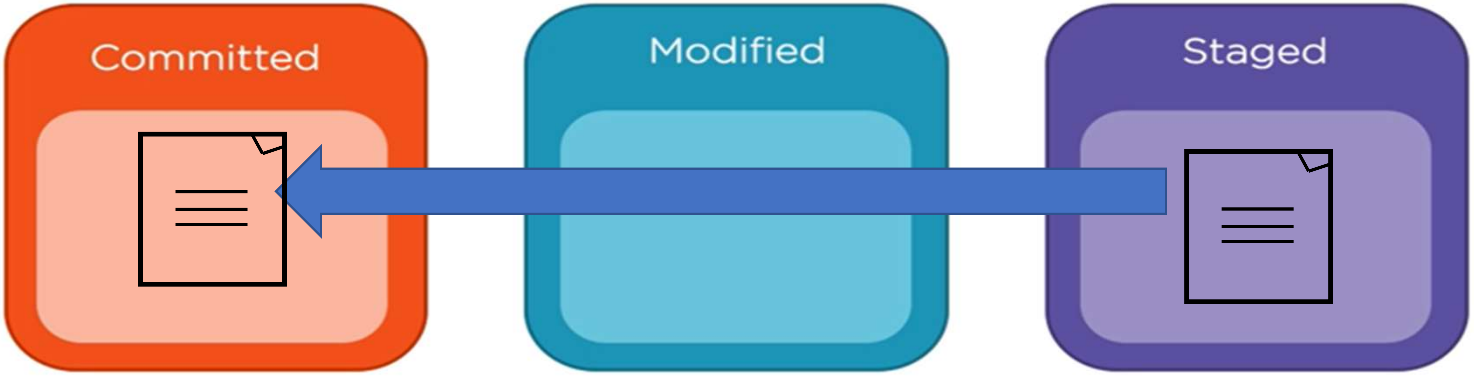
The Basics of GIT (staging)



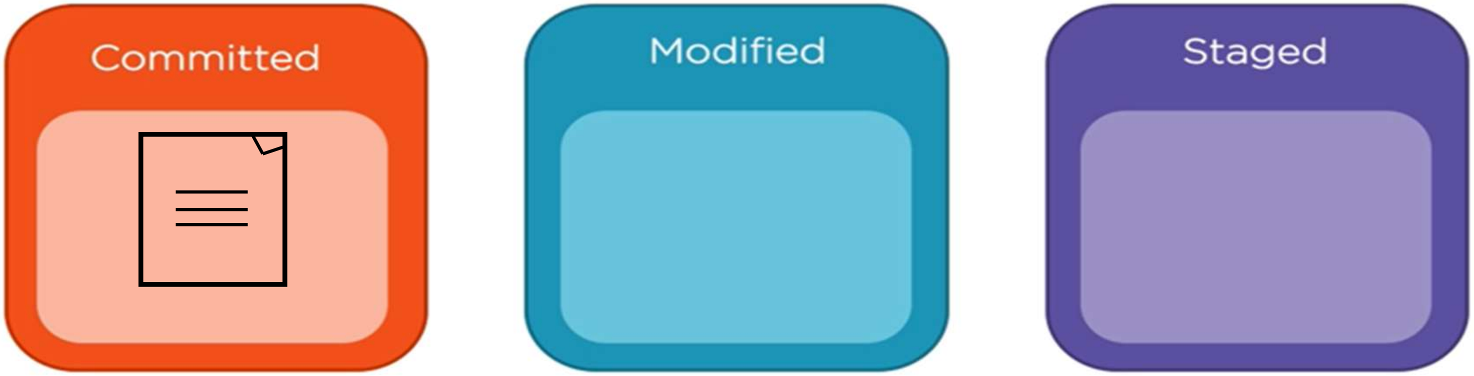
The Basics of GIT (Staging)



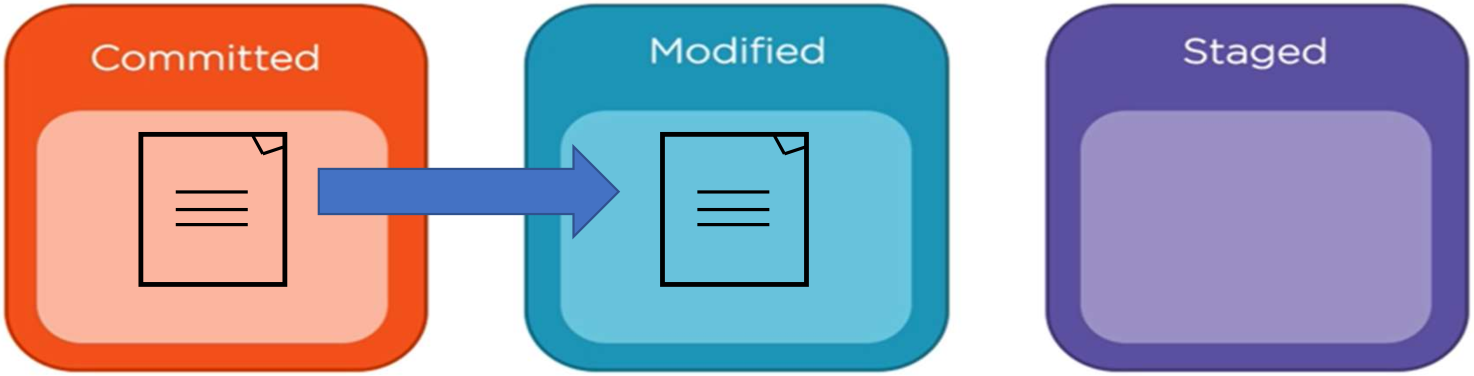
The Basics of GIT (Commit)



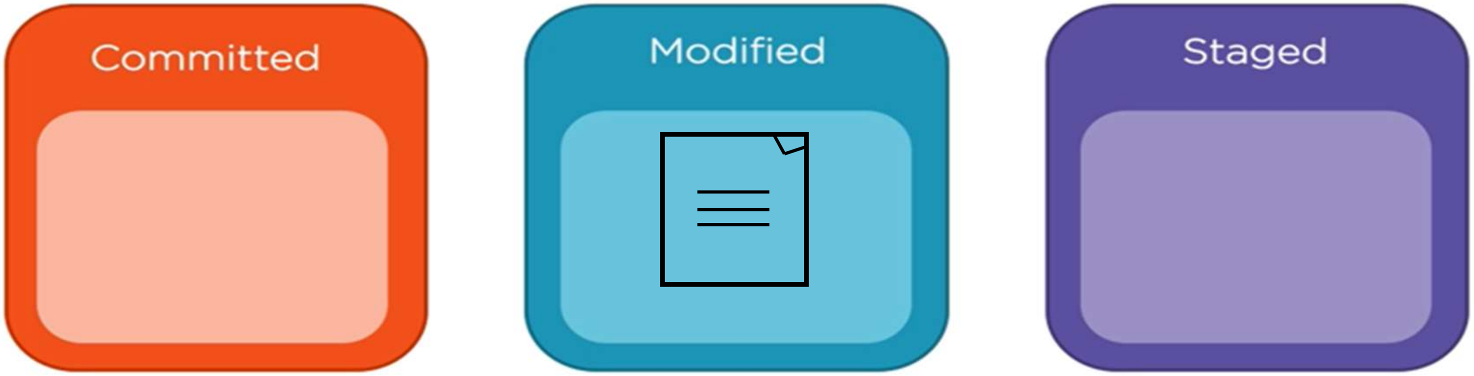
The Basics of GIT (Commit)



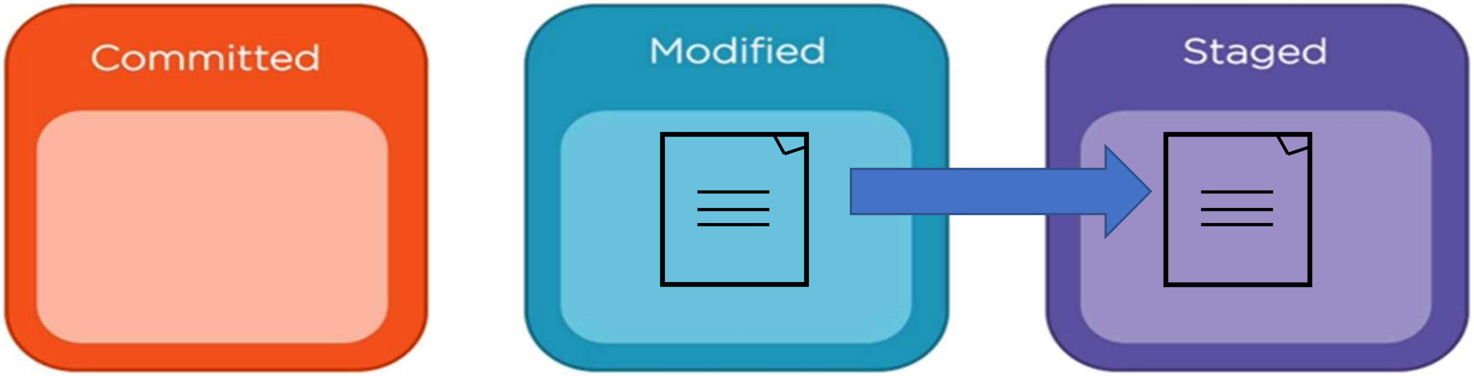
The Basics of GIT (Modifying)



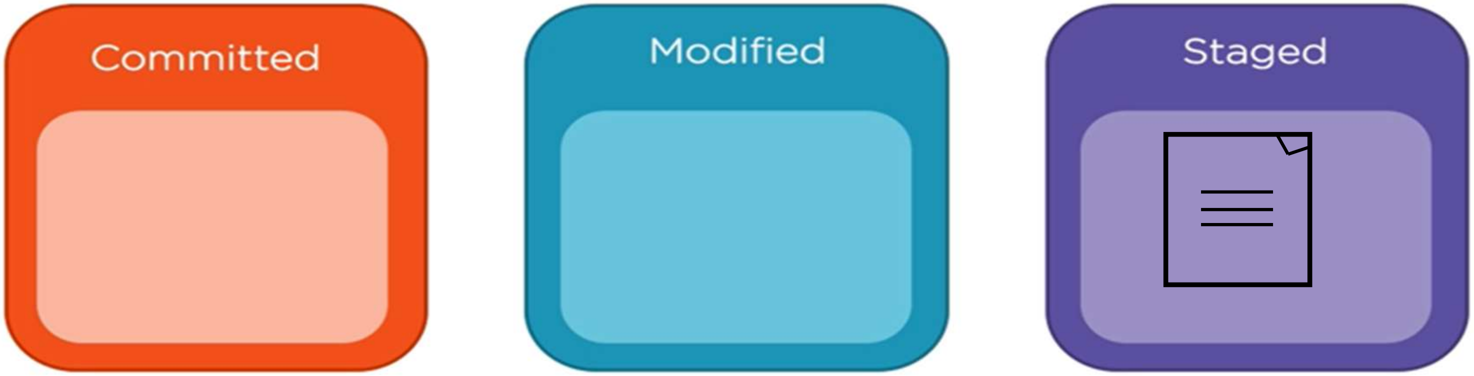
The Basics of GIT (Modifying)



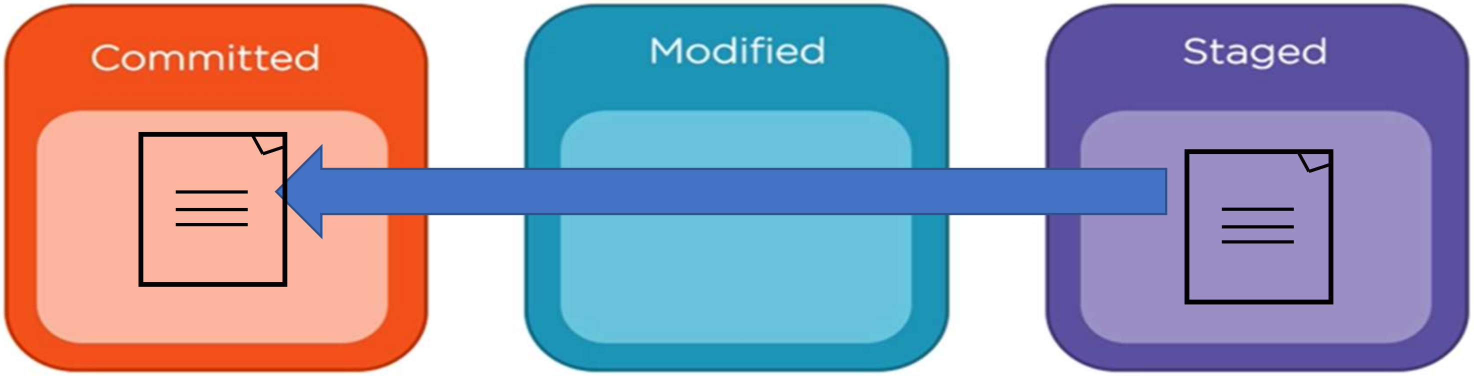
The Basics of GIT (Modifying)



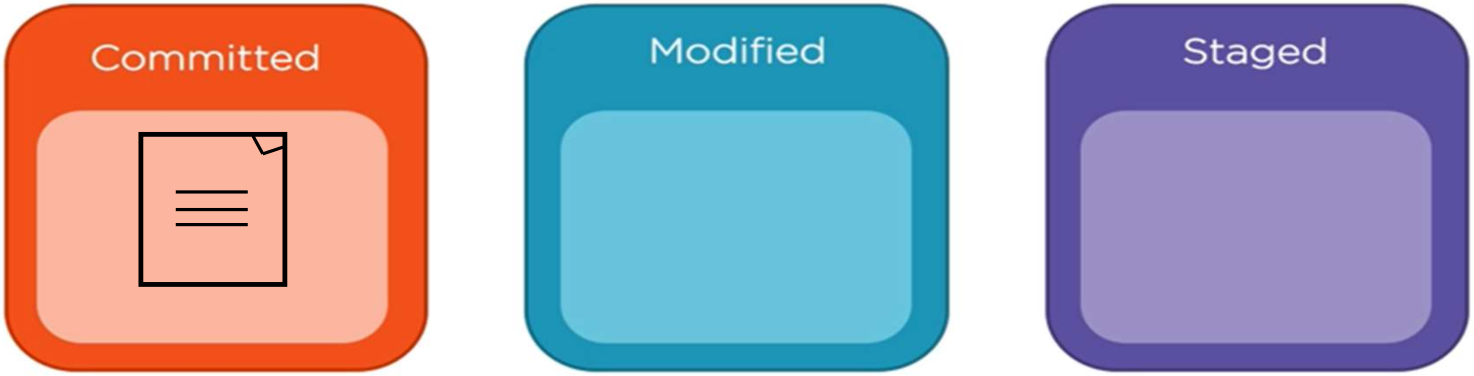
The Basics of GIT (Modifying)



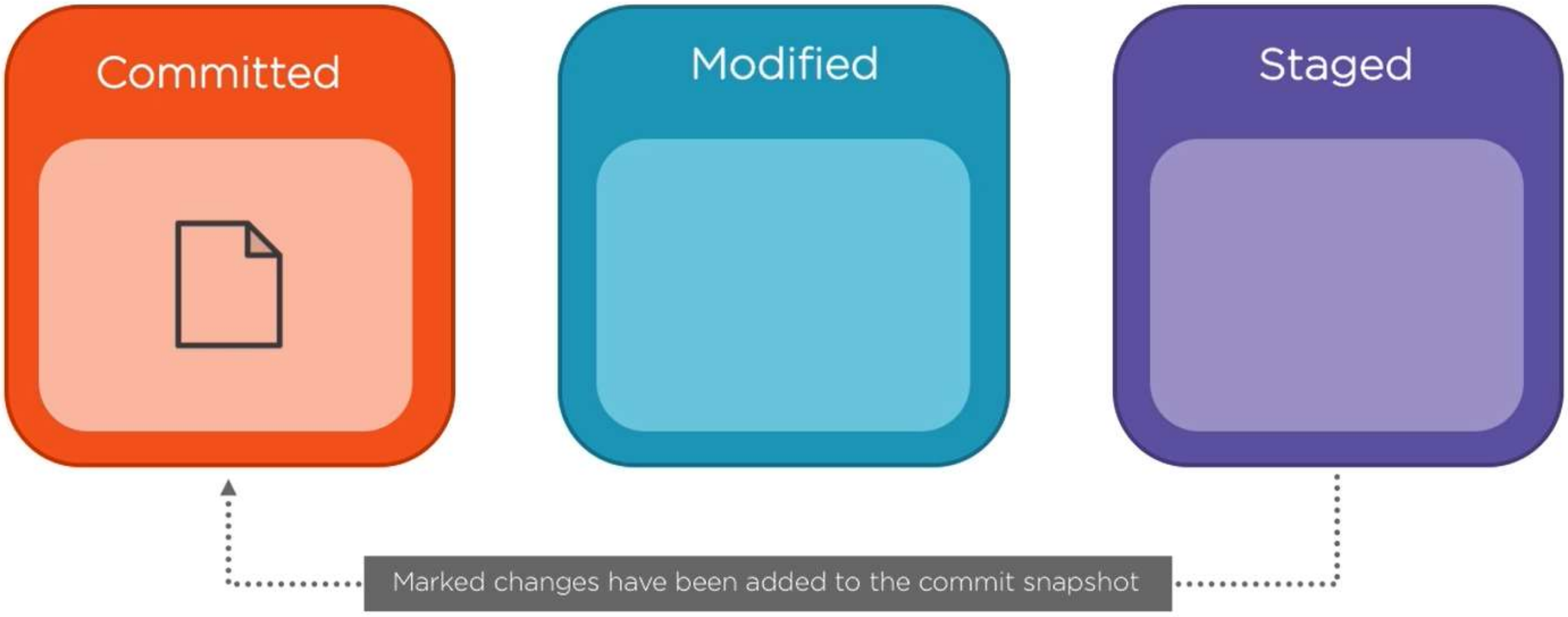
The Basics of GIT (Commit)



The Basics of GIT (Commit)



The Three Stages of a File



Committed

Modified

Staged

Marked changes have been added to the commit snapshot

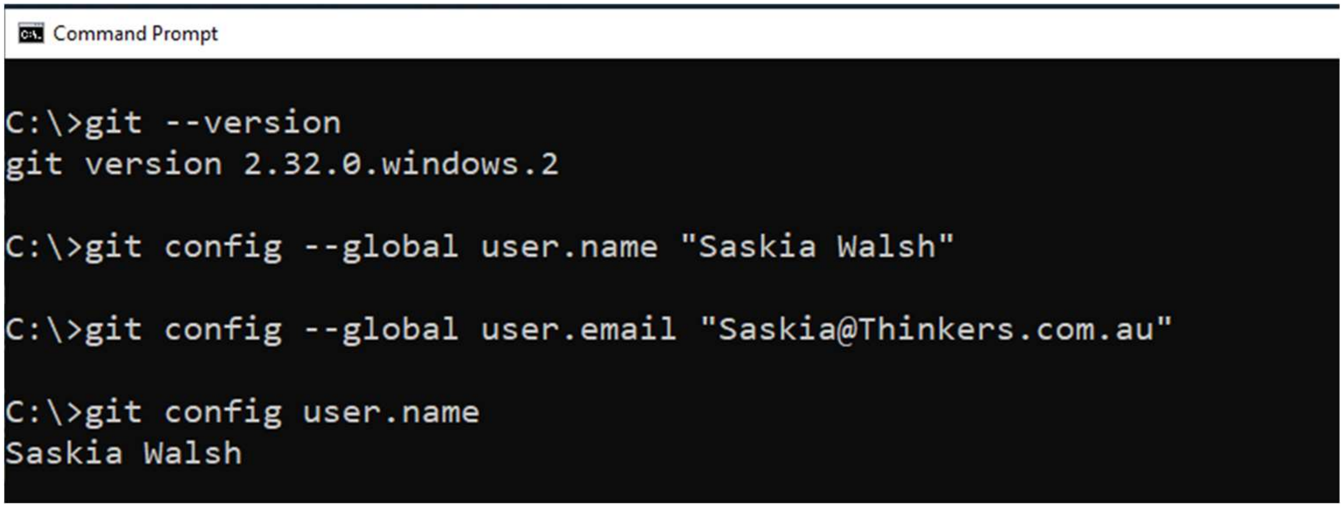
Install Git on Windows

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

Follow me

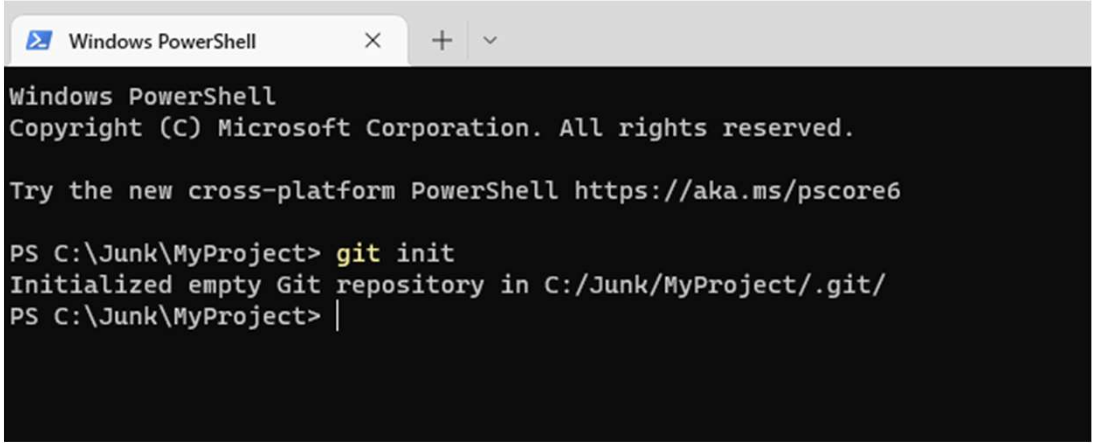
- Do what I am doing on the LAB01 virtual machine…

After Install, do some basic config



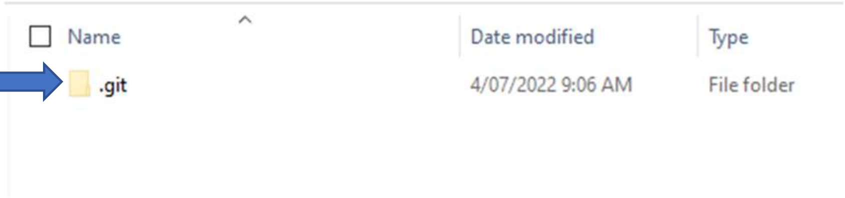
```
C:\>git --version
git version 2.32.0.windows.2

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



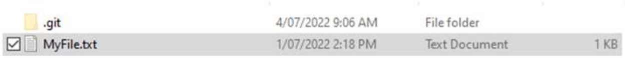
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

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

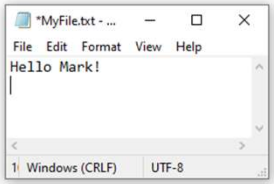


□ Name	Date modified	Type
.git	4/07/2022 9:06 AM	File folder

Adding files, staging

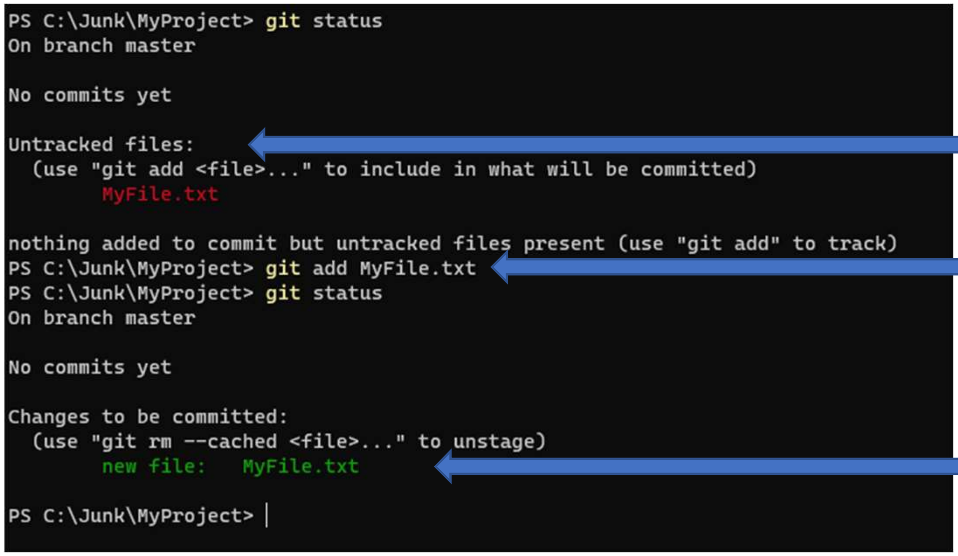


.git	4/07/2022 9:06 AM	File folder	
☑ MyFile.txt	1/07/2022 2:18 PM	Text Document	1 KB



```
Hello Mark!
```

1. add a file



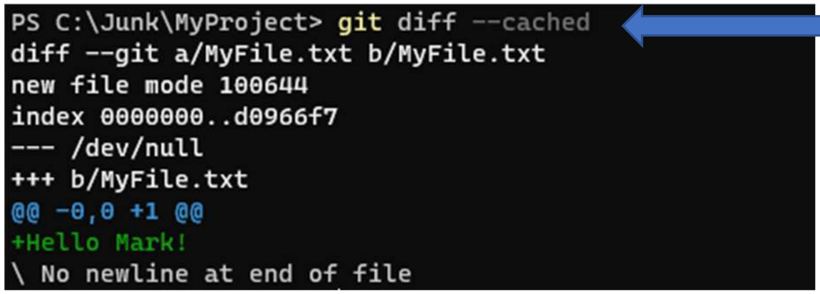
```
PS C:\Junk\MyProject> git status
On branch master

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

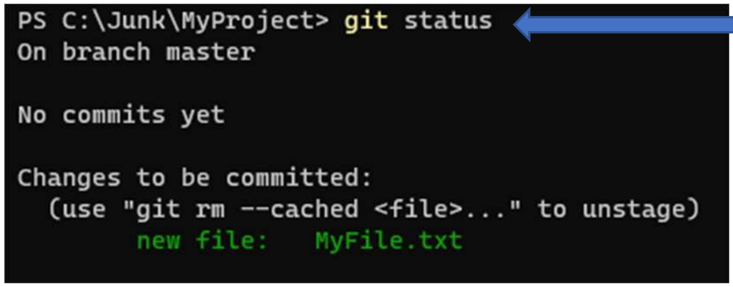
4. See the file is staged and ready to be committed

Adding files, committing



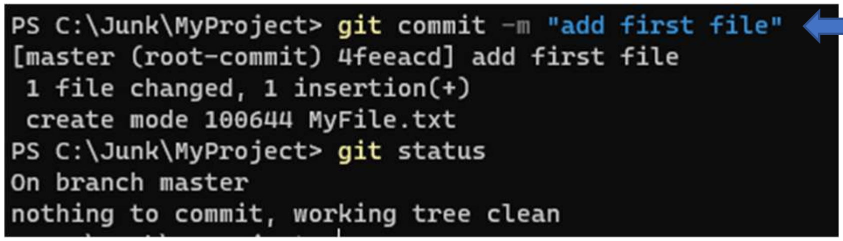
```
PS C:\Junk\MyProject> git diff --cached
diff --git a/MyFile.txt b/MyFile.txt
new file mode 100644
index 0000000..d0966f7
--- /dev/null
+++ b/MyFile.txt
@@ -0,0 +1 @@
+Hello Mark!
\ No newline at end of file
```

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
```

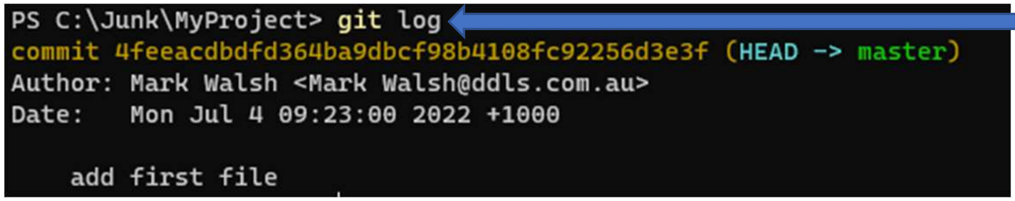
2. See what will be committed



```
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

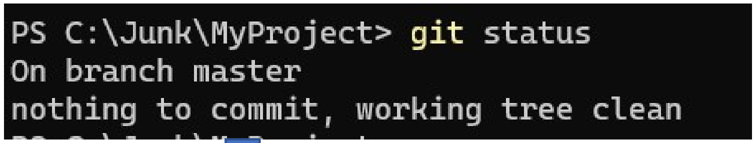
Adding files, committting



```
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
```

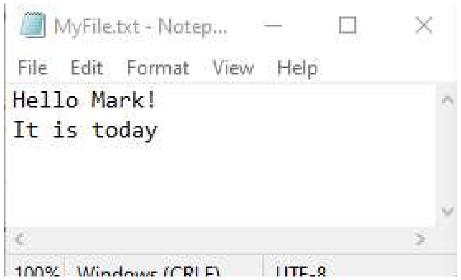
1. See History

Changing the files in a repo



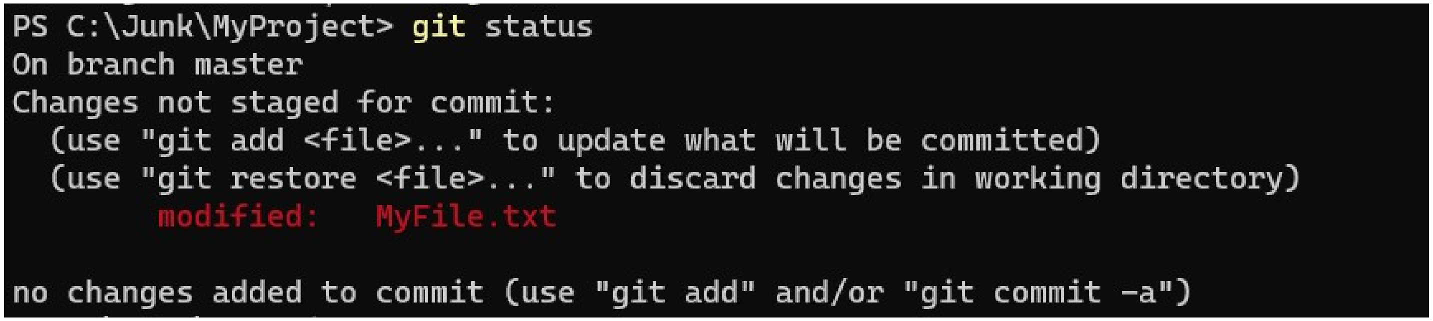
```
PS C:\Junk\MyProject> git status
On branch master
nothing to commit, working tree clean
```

Before modification



```
Hello Mark!
It is today
```

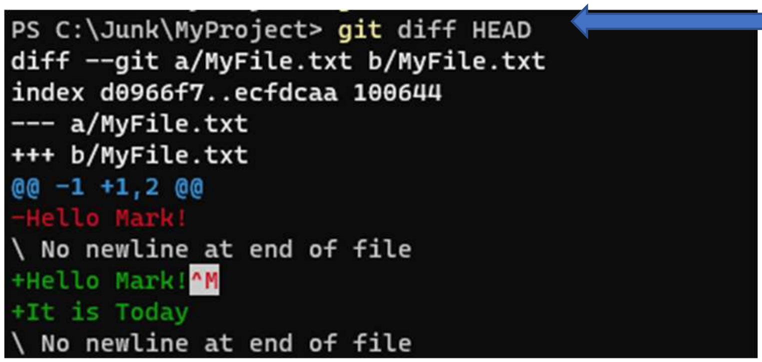
Modify the file



```
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")
```

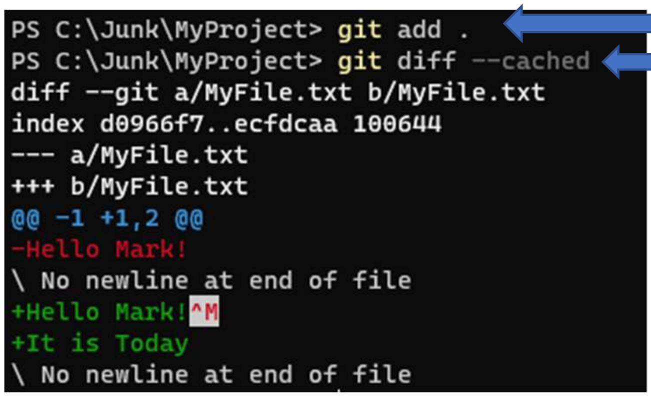
See that git knows it’s modified

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 +1,2 @@
-Hello Mark!
\ No newline at end of file
+Hello Mark!^M
+It is Today
\ No newline at end of file
```

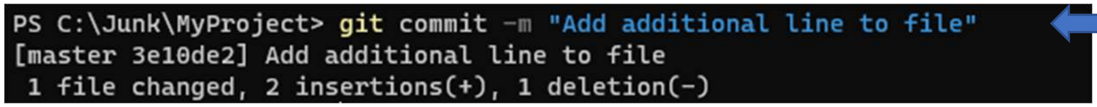
Display what is different between unstaged changes and last commit



```
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 +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

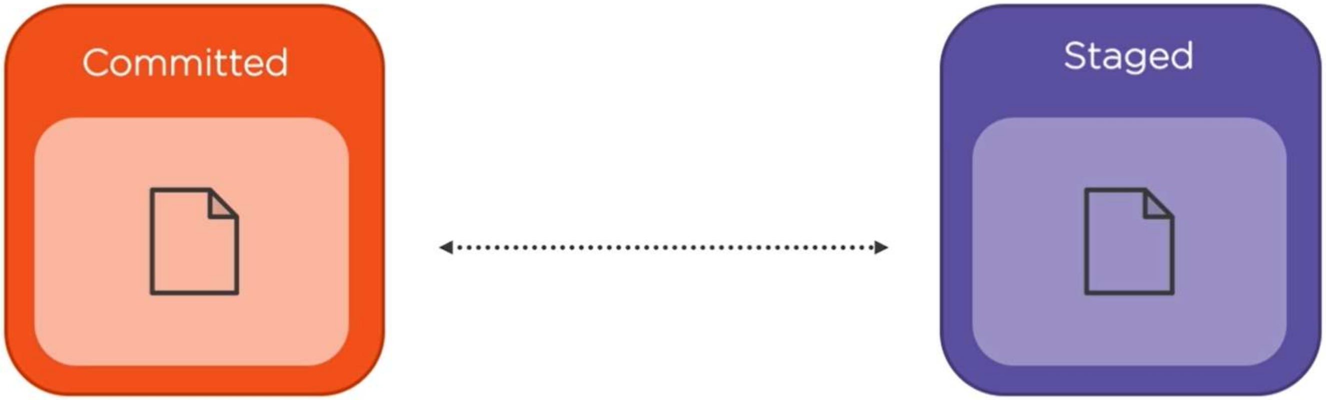


```
PS C:\Junk\MyProject> git commit -m "Add additional line to file"
[master 3e10de2] Add additional line to file
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Commit changes to the repo
(then run git log)

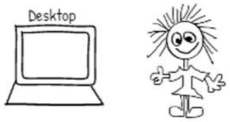
Displaying differences before doing a commit

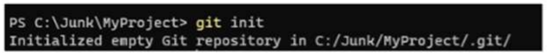
Git diff --Staged



also: git diff –staged –no-renames

Summary of git basics



1. 

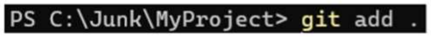
```
PS C:\Junk\MyProject> git init
Initialized empty Git repository in C:/Junk/MyProject/.git/
```

Initialise a repository

2.

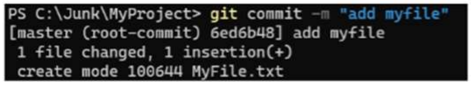
MyFile.txt

Add a file

3. 

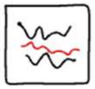
```
PS C:\Junk\MyProject> git add .
```

Stage to Commit

4. 

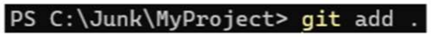
```
PS C:\Junk\MyProject> git commit -m "add myfile"
[master (root-commit) 6ed6b48] add myfile
 1 file changed, 1 insertion(+)
 create mode 100644 MyFile.txt
```

Commit

5. 

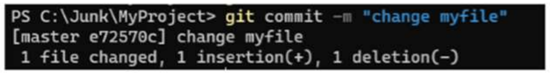
MyFile.txt

Change the file

6. 

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

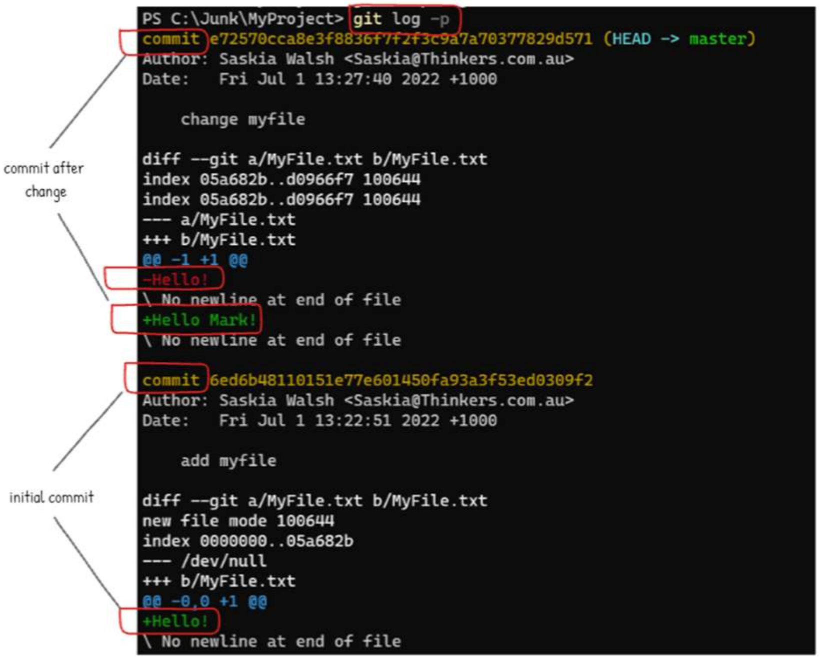
Stage to Commit

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
```

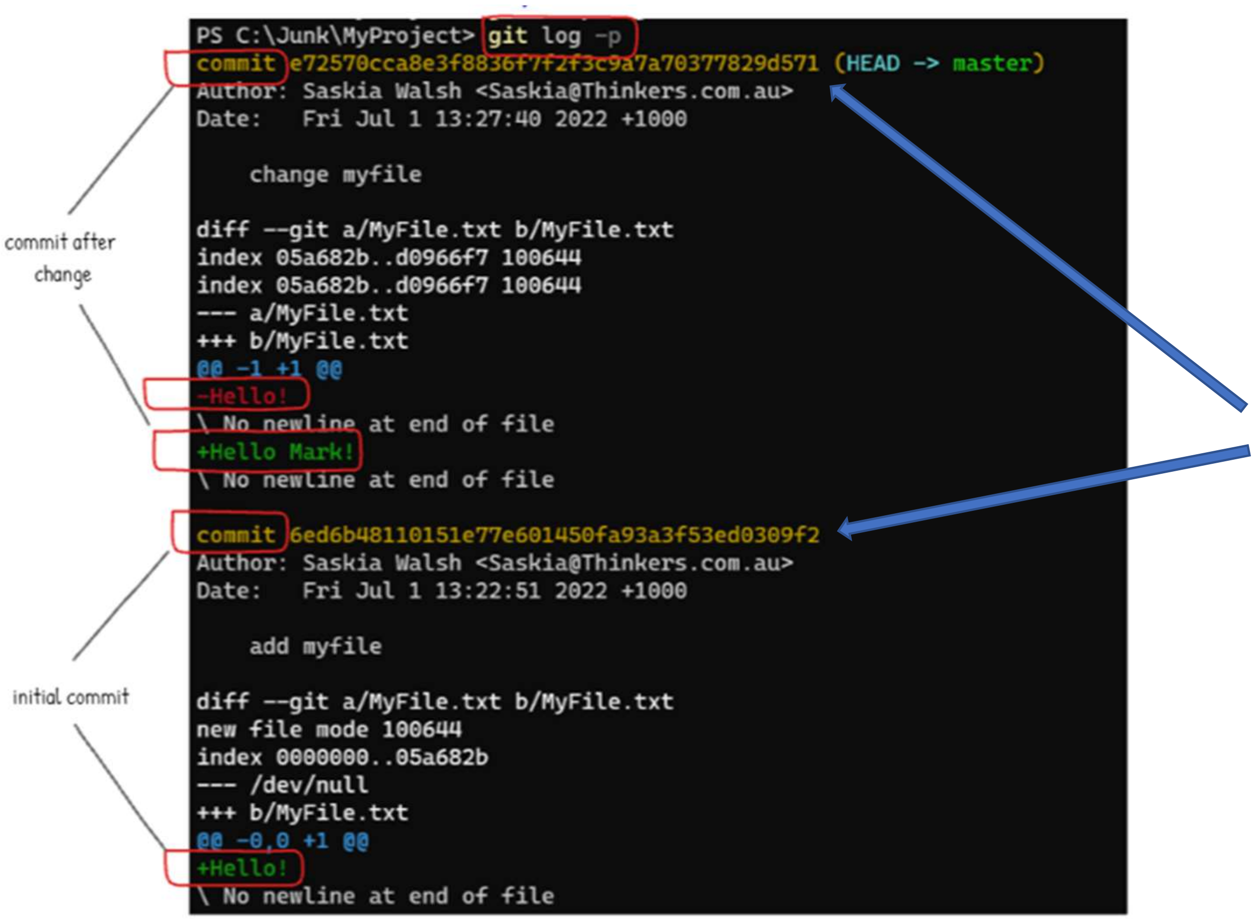
Follow me

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



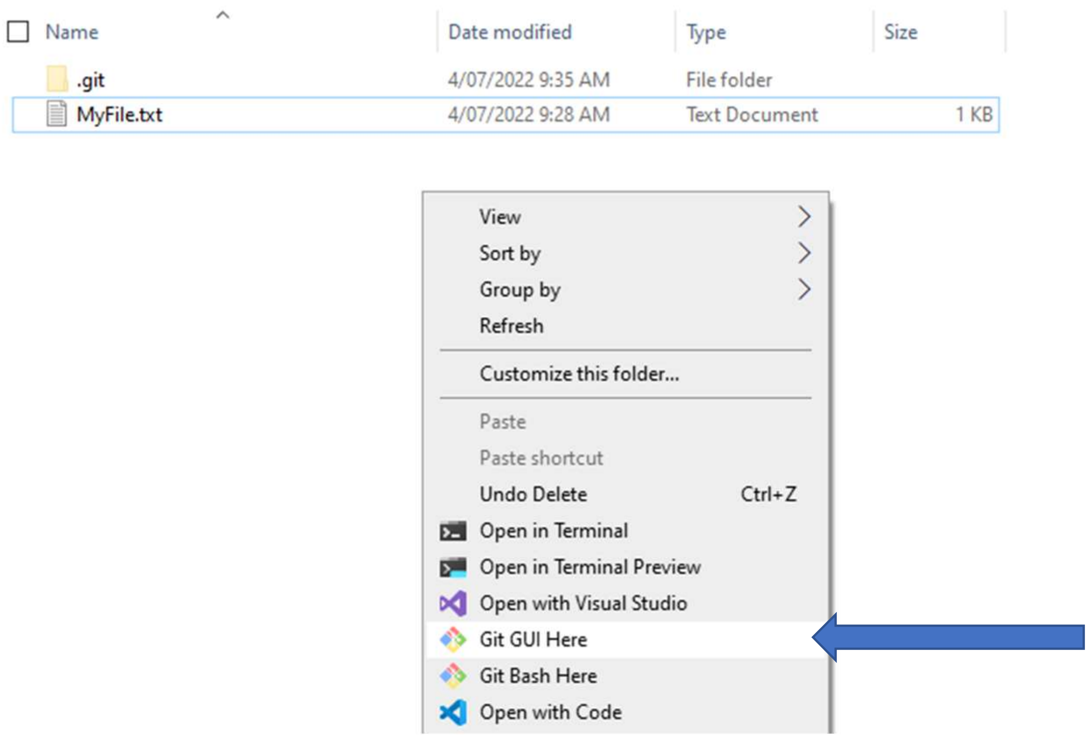
```
PS C:\Junk\MyProject> git log -p
commit e72570cca8e3f8836f7f2f3c9a7a70377829d571 (HEAD -> master)
Author: Saskia Walsh <Saskia@Thinkers.com.au>
Date:   Fri Jul 1 13:27:40 2022 +1000

    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
```

What is that long series of characters?

Follow me

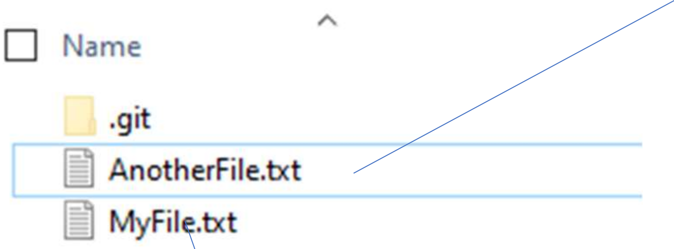
- Do what I am doing on the LAB01 virtual machine…

How about some kind of graphical tool? GitGui



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

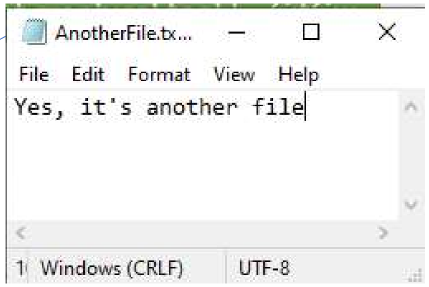


Name

.git

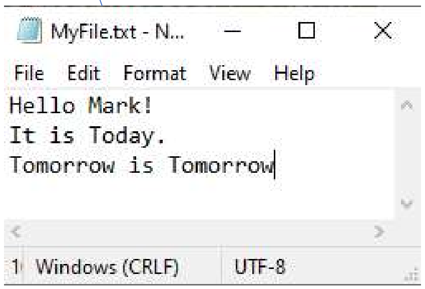
AnotherFile.txt

MyFile.txt



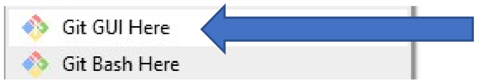
```
Yes, it's another file
```

1. Add a file



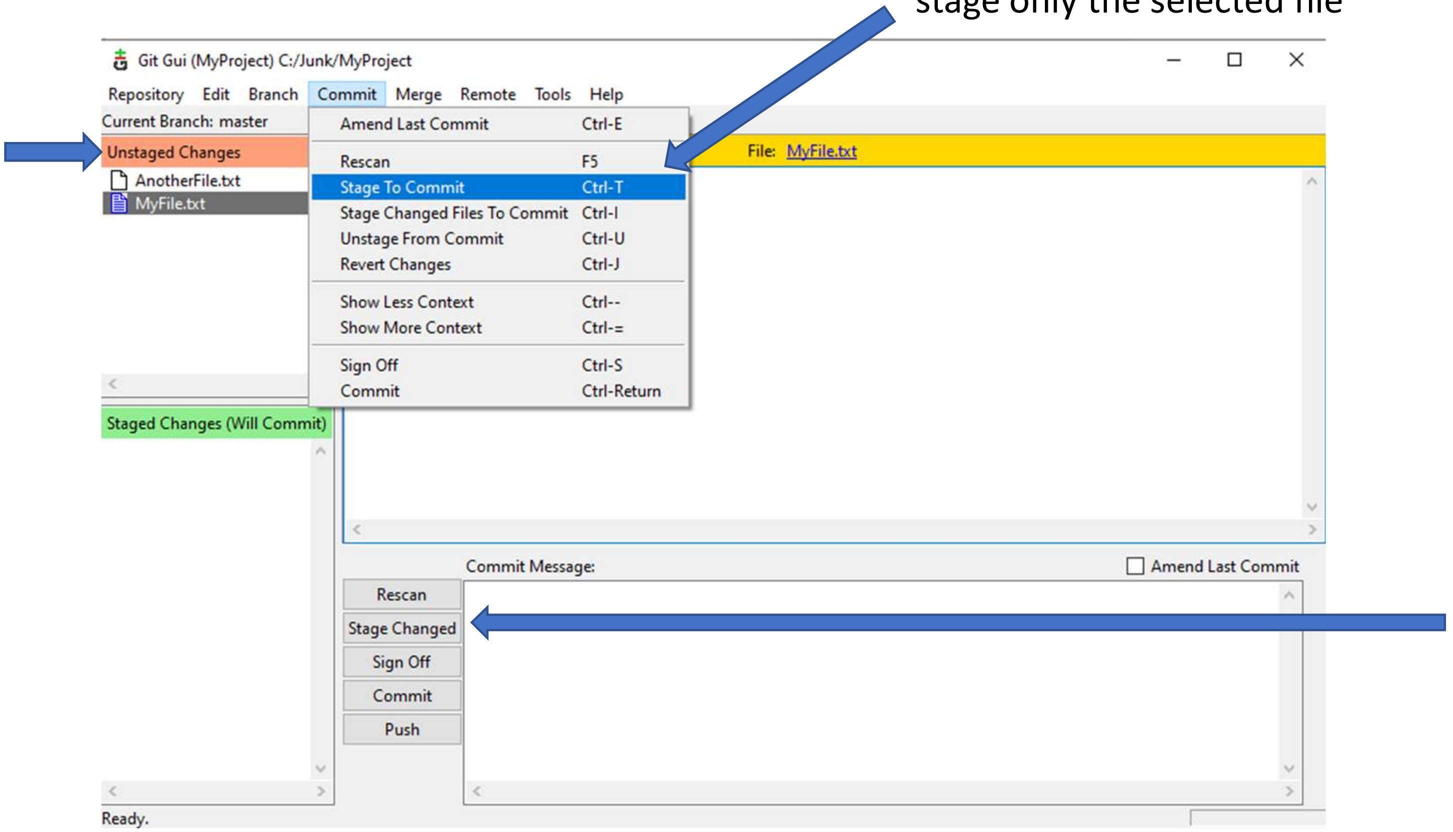
```
Hello Mark!
It is Today.
Tomorrow is Tomorrow
```

2. Change a file



3. Open GitGui

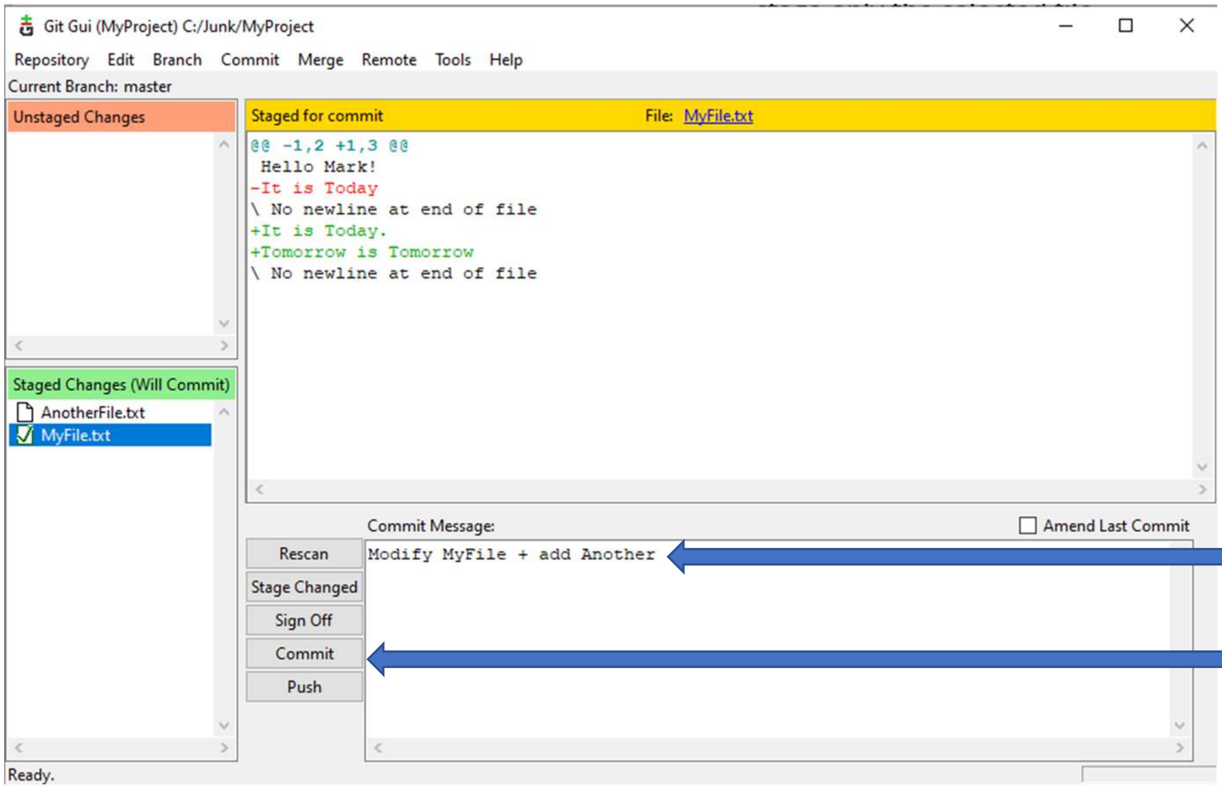
How about some kind of graphical tool? GitGui



stage only the selected file

stage all files

How about some kind of graphical tool? GitGui



```
@@ -1,2 +1,3 @@
 Hello Mark!
-It is Today
\ No newline at end of file
+It is Today.
+Tomorrow is Tomorrow
\ No newline at end of file
```

Commit Message: Modify MyFile + add Another

The Commit message

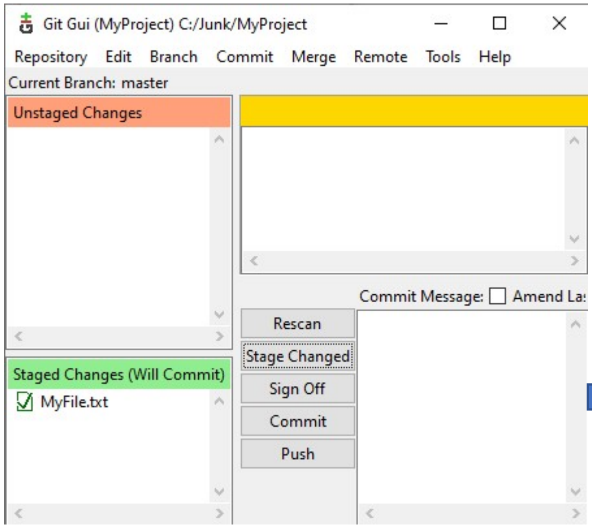
Commit all staged files

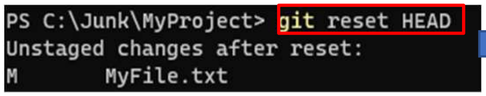
A useful git command

```
git checkout .

git checkout filename.suffix
```

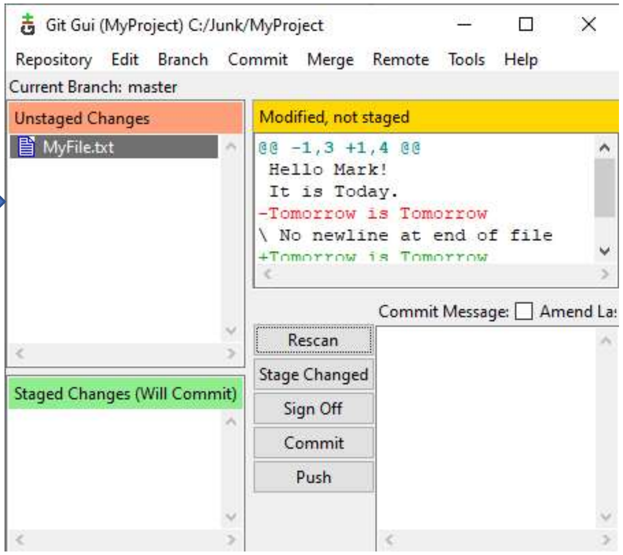
If you have changed files but not yet run ***git add***, you can undo your local changes with ***git checkout***

How do I *unstage* all staged files?



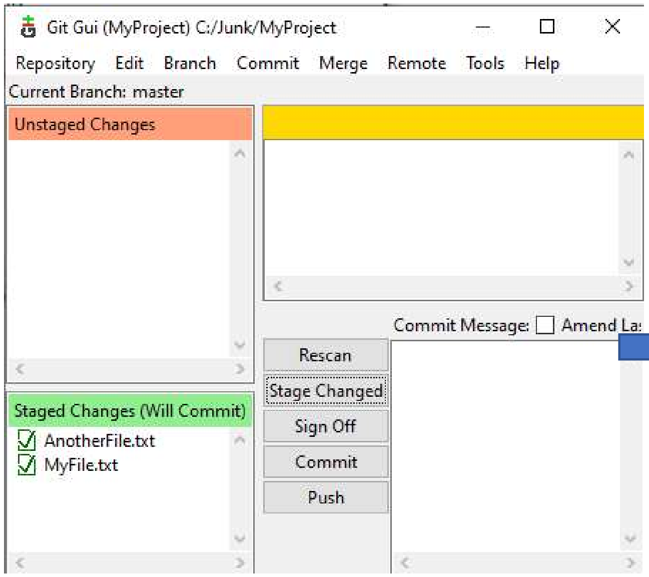


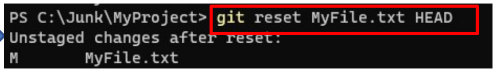
```
PS C:\Junk\MyProject> git reset HEAD
Unstaged changes after reset:
M       MyFile.txt
```



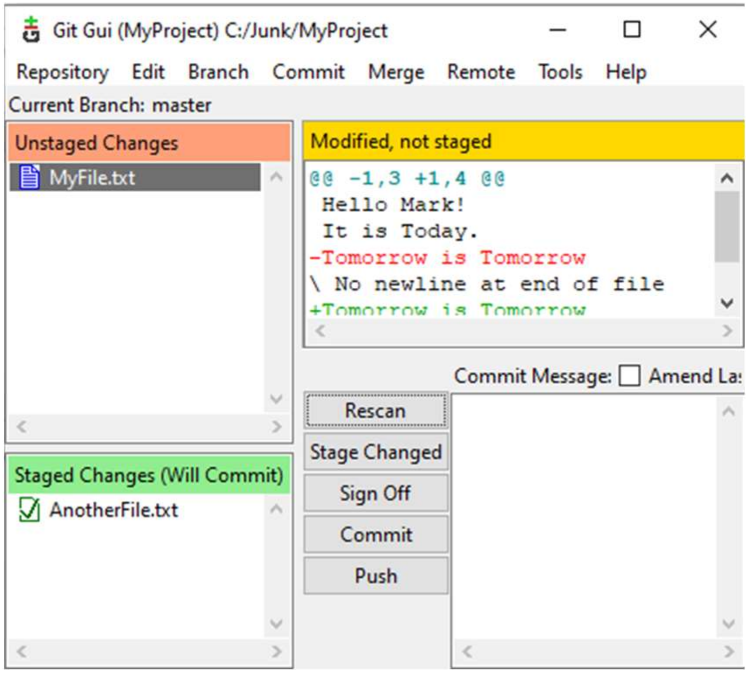
```
@@ -1,3 +1,4 @@
 Hello Mark!
 It is Today.
-Tomorrow is Tomorrow
\ No newline at end of file
+Tomorrow is Tomorrow
```

How do I *unstage* one staged file?



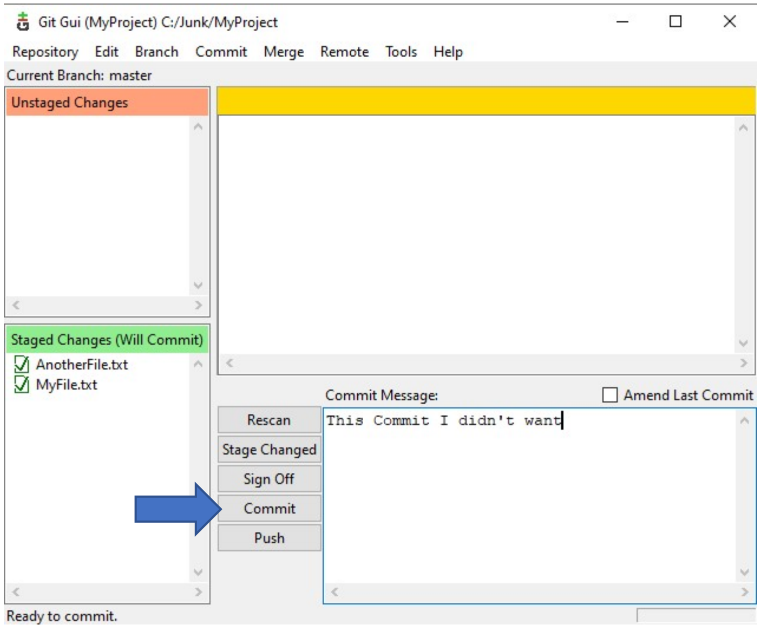


```
PS C:\Junk\MyProject> git reset MyFile.txt HEAD
Unstaged changes after reset:
M       MyFile.txt
```



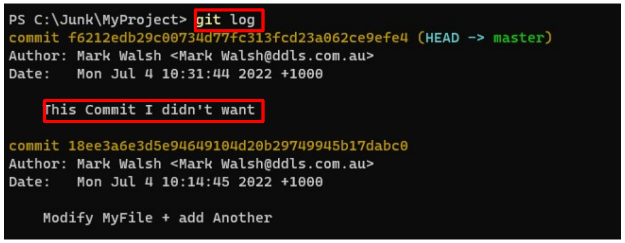
```
@@ -1,3 +1,4 @@
 Hello Mark!
 It is Today.
-Tomorrow is Tomorrow
\ No newline at end of file
+Tomorrow is Tomorrow
```

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

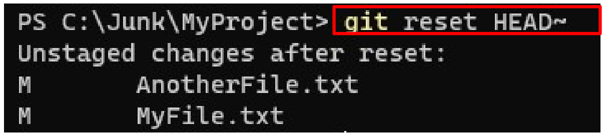


Commit Message:

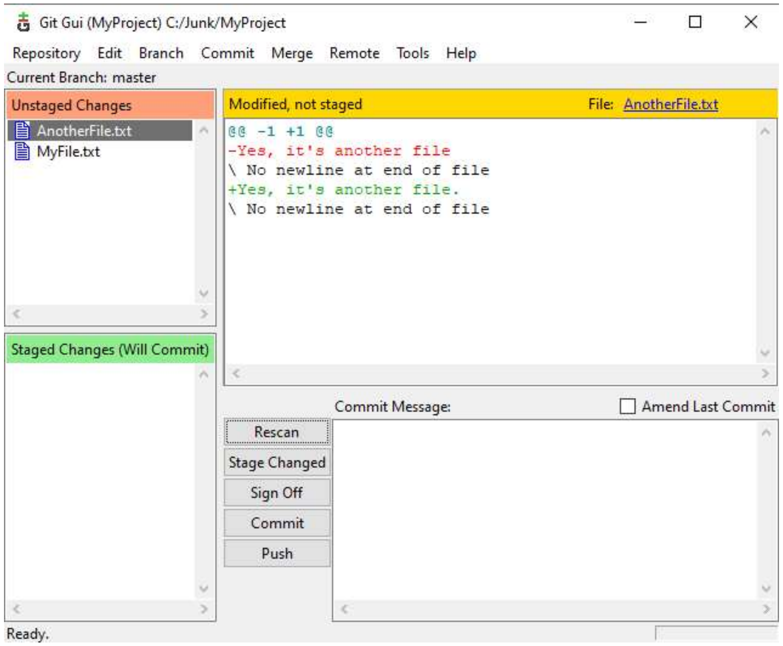
```
This Commit I didn't want
```



```
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
```



```
@@ -1 +1 @@
-Yes, it's another file
\ No newline at end of file
+Yes, it's another file.
\ No newline at end of file
```

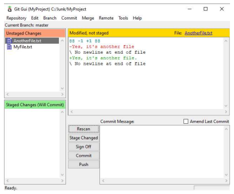
git reset

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*



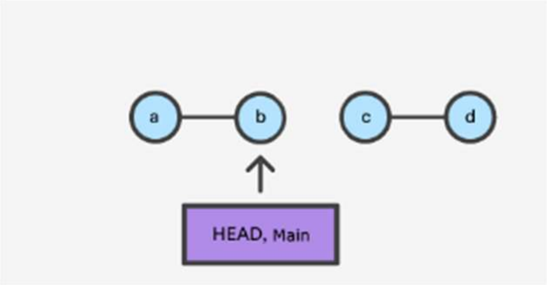
Current Branch: master

Modified, not staged File: AnotherFile.txt

```
@@ -1 +1 @@
-Yes, it's another file
\ No newline at end of file
+Yes, it's another file.
\ No newline at end of file
```

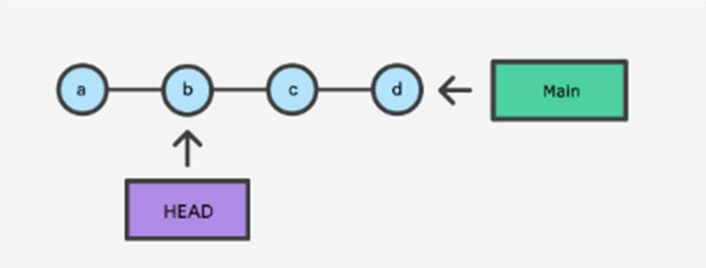
git reset vs git checkout

git reset 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**

git checkout b



Do that again?

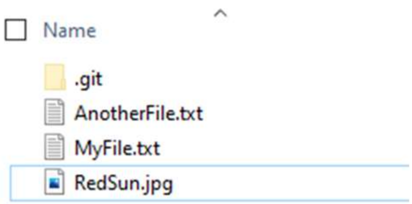
1. Make a change to a file
2. Undo the changes *(git checkout .)*

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.



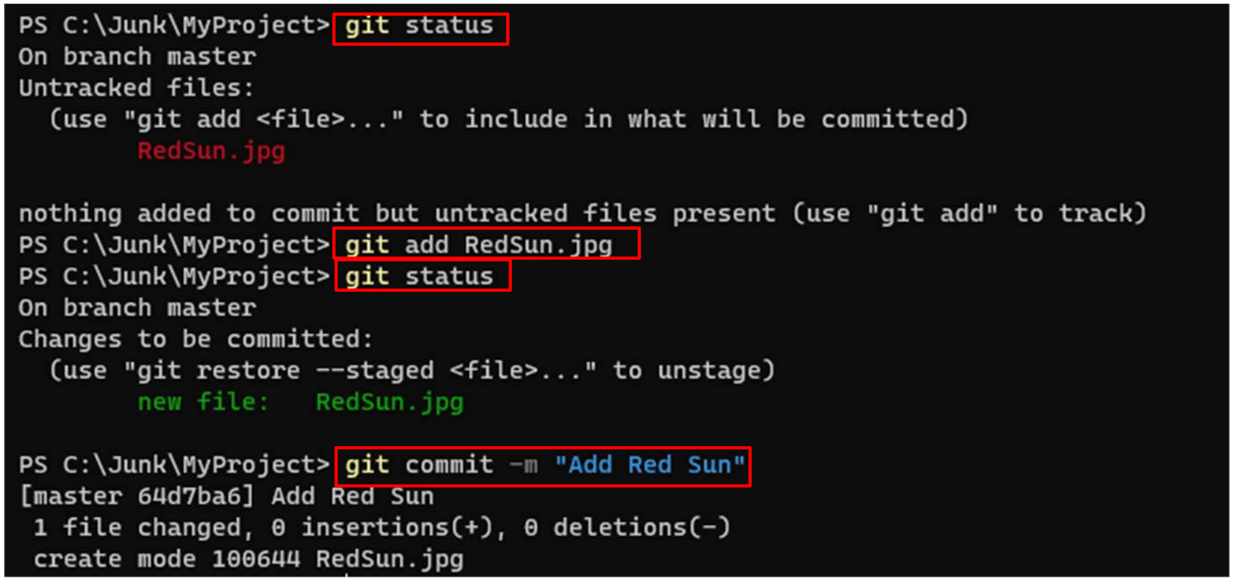
□ Name

.git

AnotherFile.txt

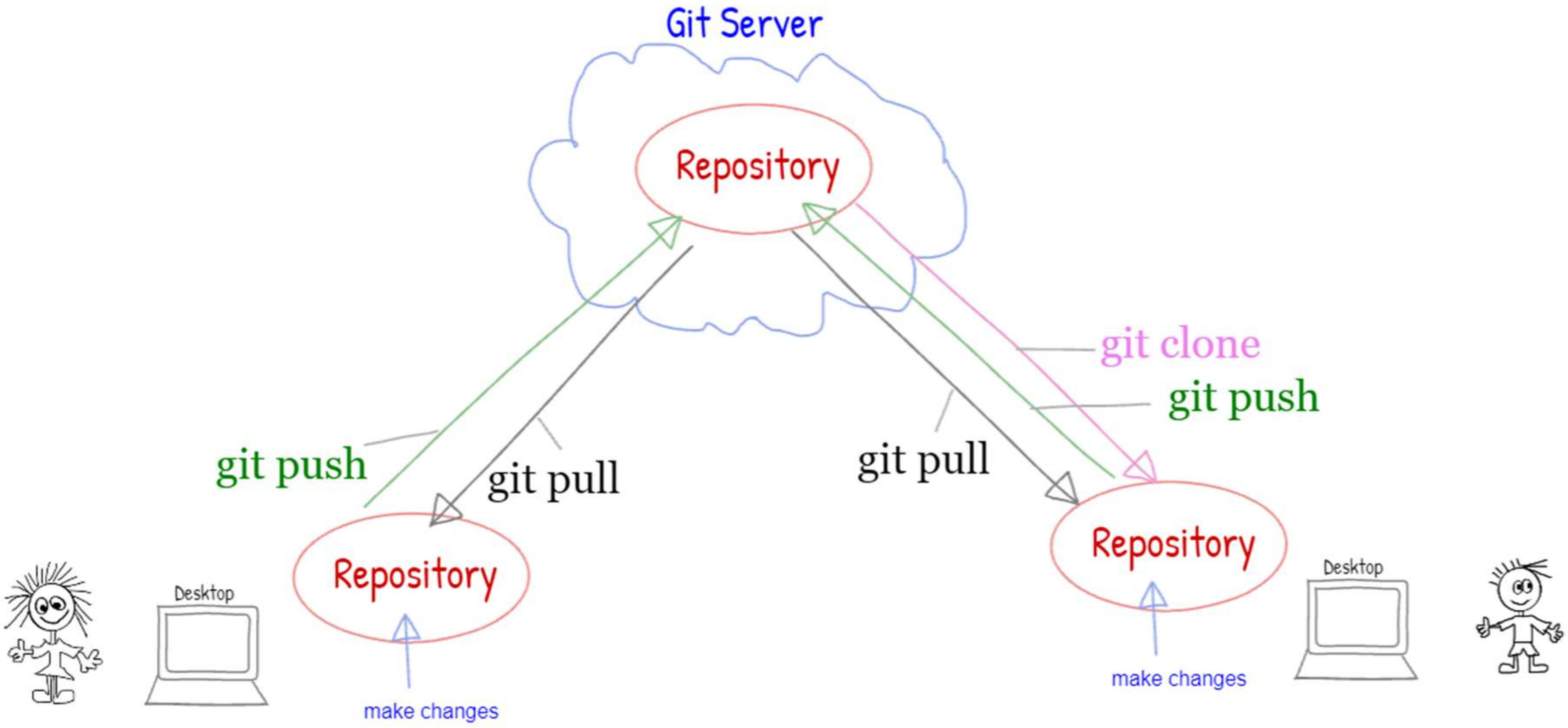
MyFile.txt

RedSun.jpg



```
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.

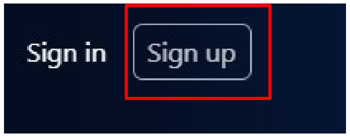
Follow me (using Lab01 VM)

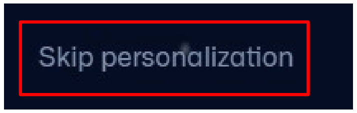
Overview:

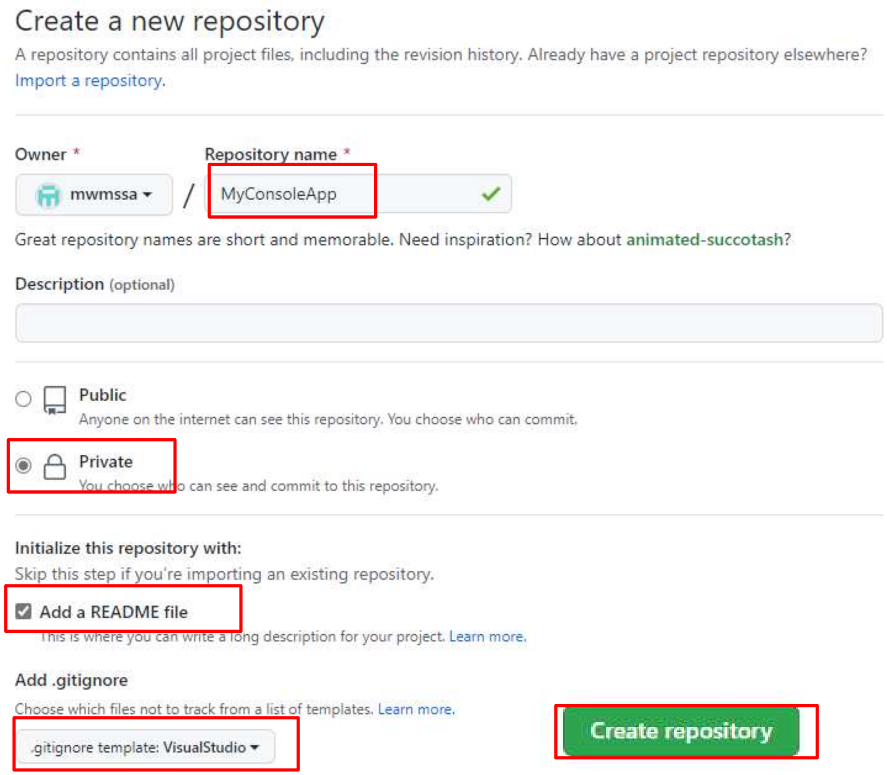
1. Create a new repo in Github (include gitignore and readme)
2. Git Clone
3. Add your application code
4. Stage files
5. Commit files
6. git push

Cloning repo's and collaborating

https://github.com/







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.

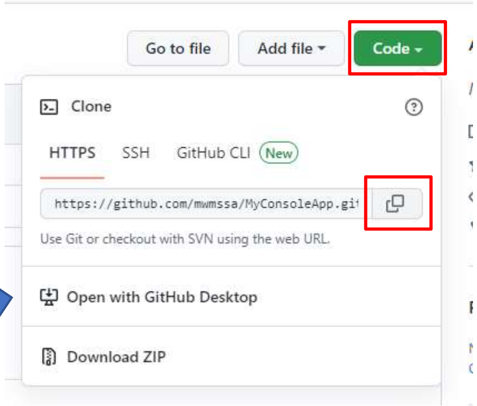
● Private
You choose who can see and commit to this repository.

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.

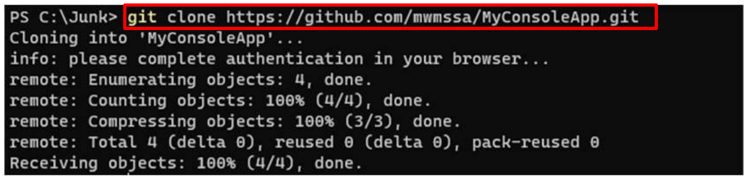
Add .gitignore
Choose which files not to track from a list of templates. Learn more.

.gitignore template: VisualStudio

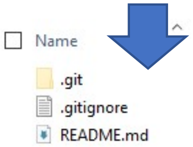


https://github.com/mwmssa/MyConsoleApp.git

Use Git or checkout with SVN using the web URL.

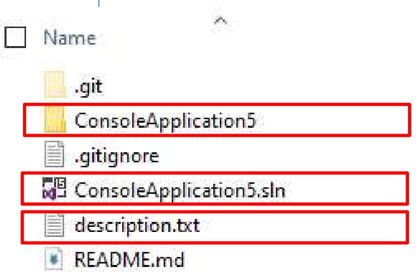


```
PS C:\Junk> git clone https://github.com/mwmssa/MyConsoleApp.git
Cloning into 'MyConsoleApp'...
info: please complete authentication in your browser...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4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



□ Name
.git
.gitignore
README.md

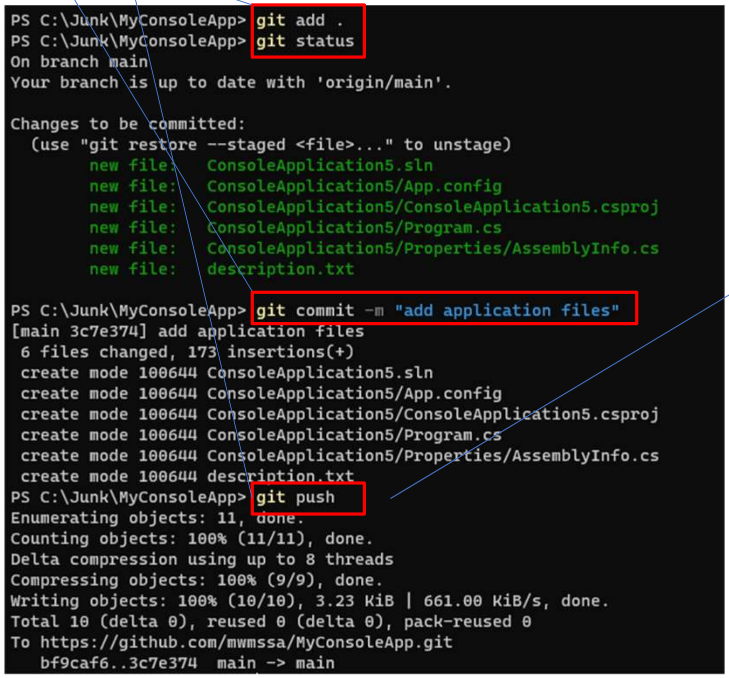
Create Application -> Stage -> Commit -> Push



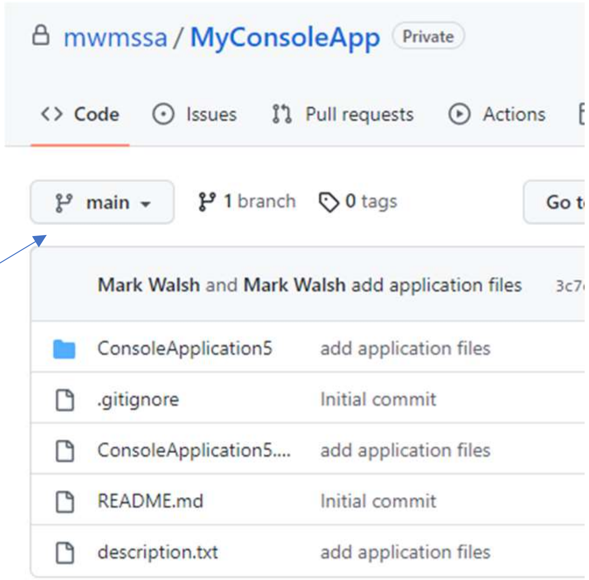
□ Name

.git
ConsoleApplication5
.gitignore
ConsoleApplication5.sln
description.txt
README.md

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
```



Mark Walsh and Mark Walsh add application files 3c7

ConsoleApplication5	add application files
.gitignore	Initial commit
ConsoleApplication5....	add application files
README.md	Initial commit
description.txt	add application files

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.

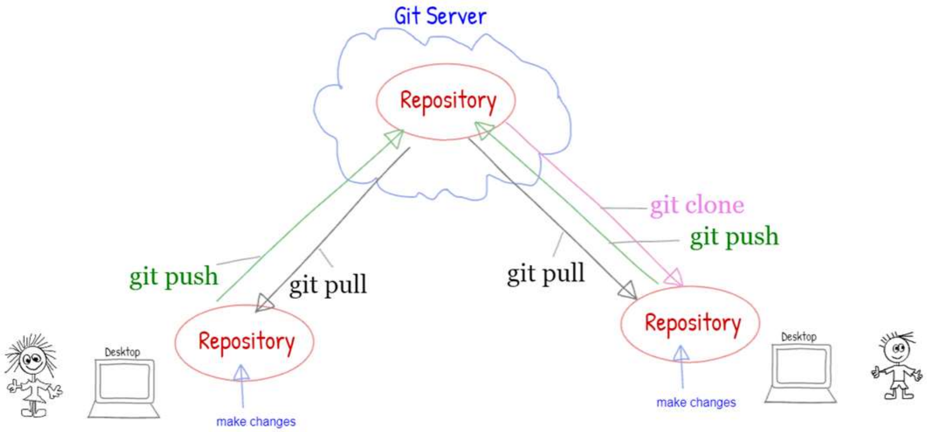
4. Git clone the repo.

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.



git

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

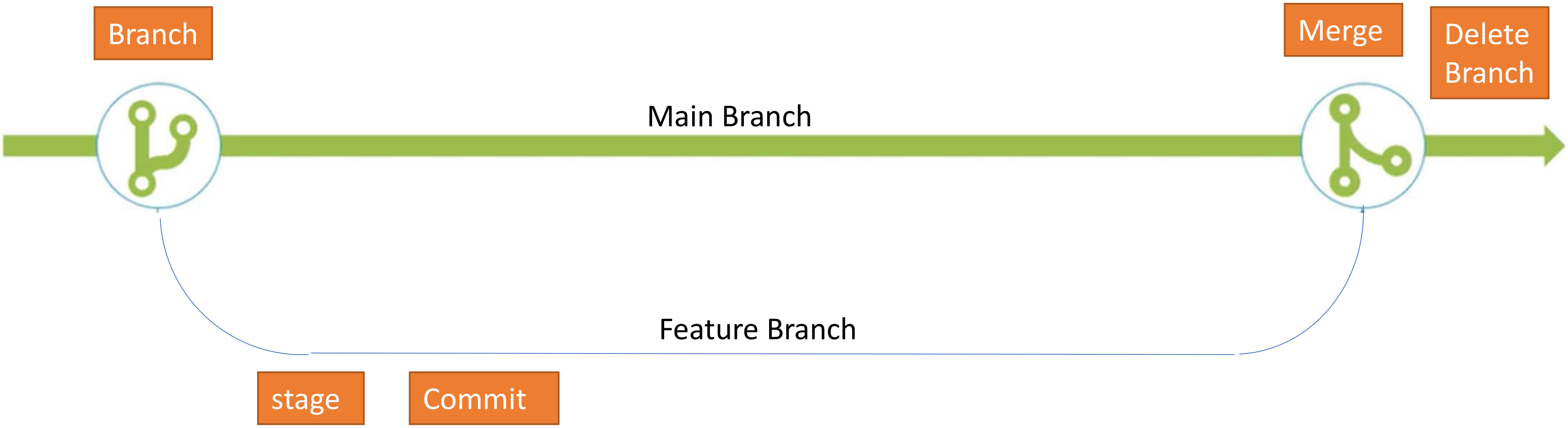
- Do this:
  1. Create a new *Public* repo in Github.
  2. Save all your work in your public repo.
  3. We can review each other’s work and help each other.

MORE

git

by Mark Walsh (AIICT)

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



Branch

Main Branch

Merge

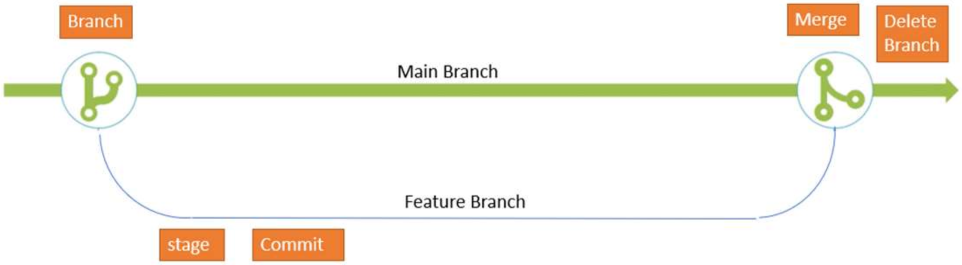
Delete Branch

Feature Branch

stage

Commit

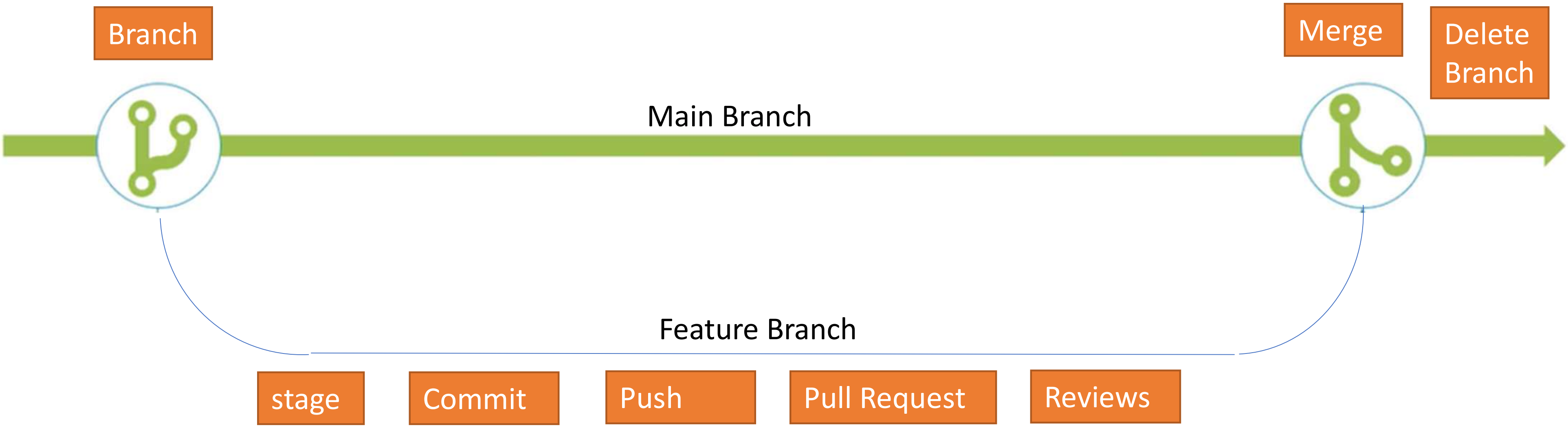
Branching on your local repo



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 git stage -> commit
9. Make some changes to the files.
10. git 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


Branch

Merge

Delete Branch

Main Branch

Feature Branch

stage

Commit

Push

Pull Request

Reviews

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.

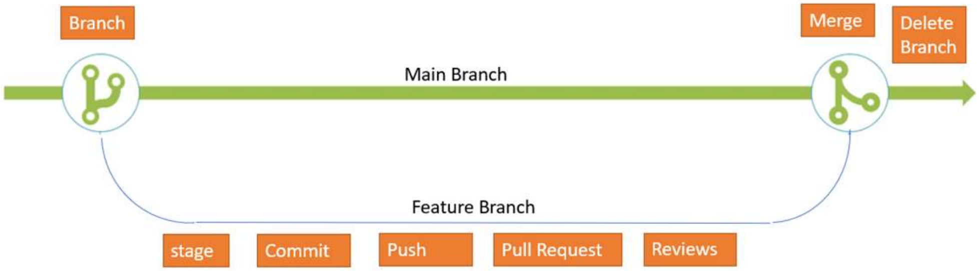
5. Checkout your feature branch.

6. Change the application code in your local repo and test it.

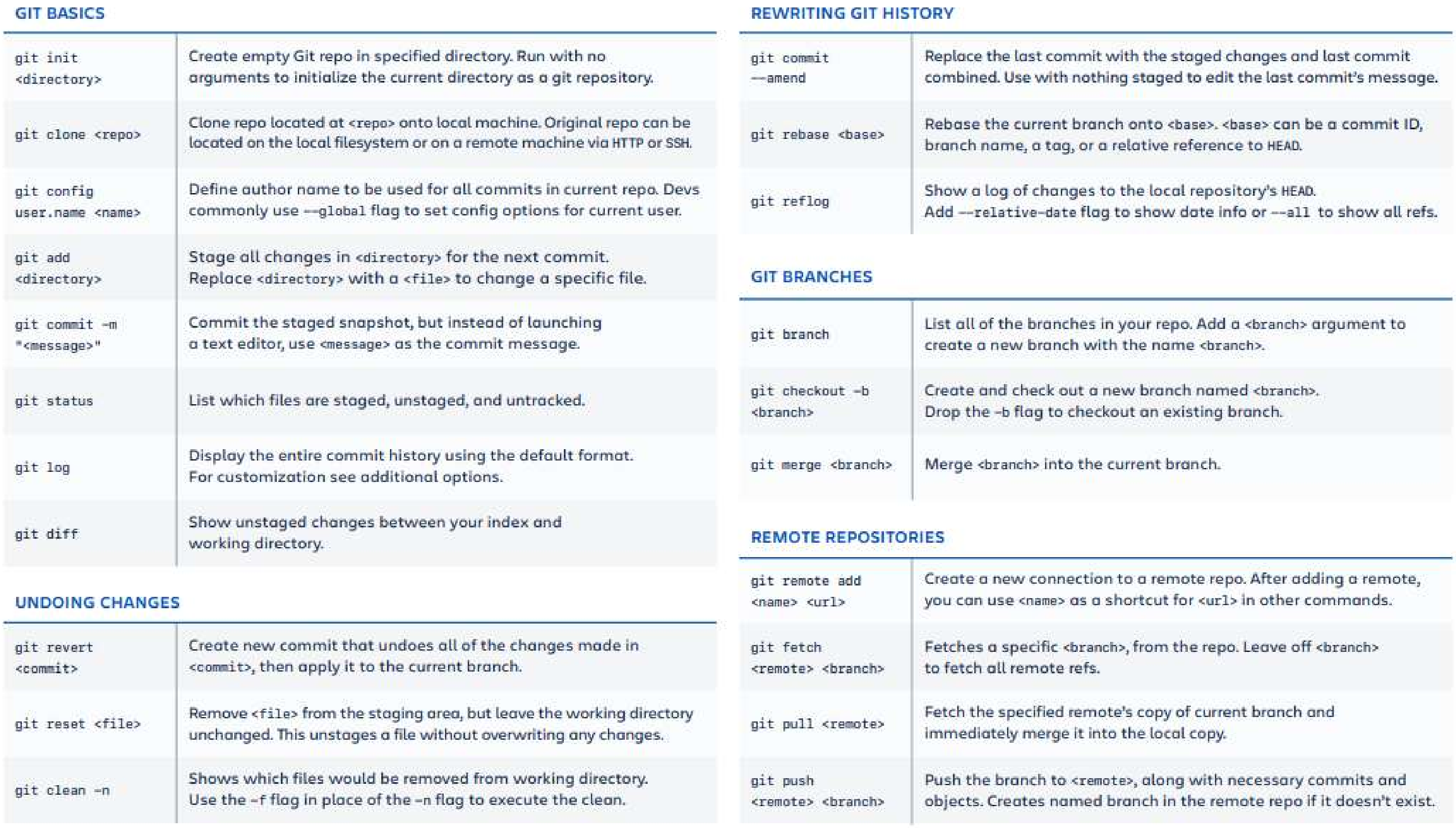
7. git stage -> commit on the feature branch

8. Push the feature branch to the origin

9. Raise a PR



Summary of key Git Commands



GIT BASICS

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

UNDOING CHANGES

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

REWRITING GIT HISTORY

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

GIT BRANCHES

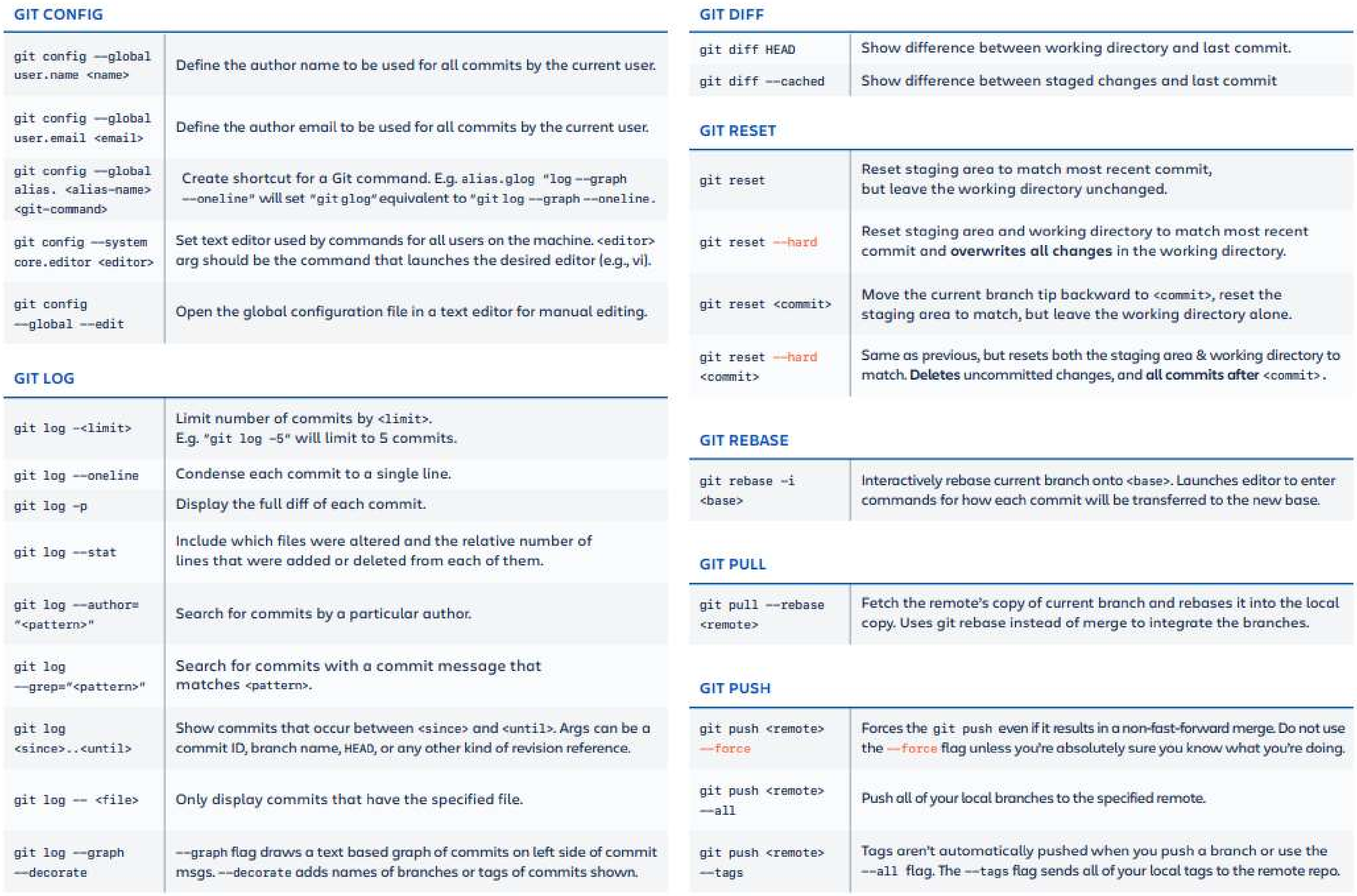
`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the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

REMOTE REPOSITORIES

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

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

Summary of key Git Commands



GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline`.
`git config --system core.editor <editor>`	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).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

GIT LOG

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

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

GIT RESET

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

GIT REBASE

`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

GIT PULL

`git pull --rebase <remote>`	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

`git push <remote> --force`	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.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	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

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!!!

X

to add:

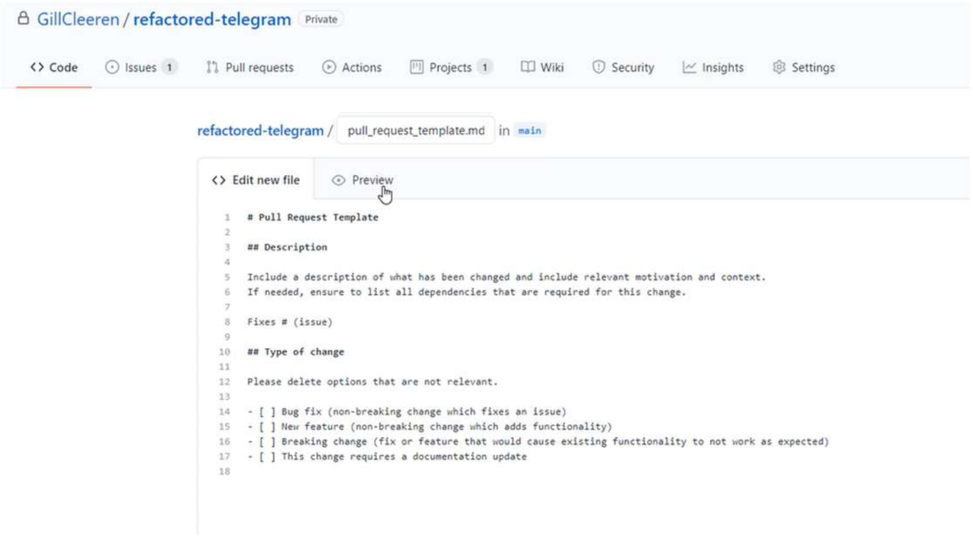
- Forks
- Public Vs Private Repos
- Understanding github screens
  - History
  - Making changes
  - Markdown
- Branching
  - Creating a branch on GitHub
    - Make changes
    - Merge back to main
  - Create branch locally
    - git status
    - git branch feature123
    - git checkout feature123
    - git add …..
    - git commit ….
    - git push –u origin feature123 (-u created the tracking relationship)

X

to add:

- Merging
  - fast forward
  - rebase
  - merge conflicts

- Show how to create a PR Template in GitHUB

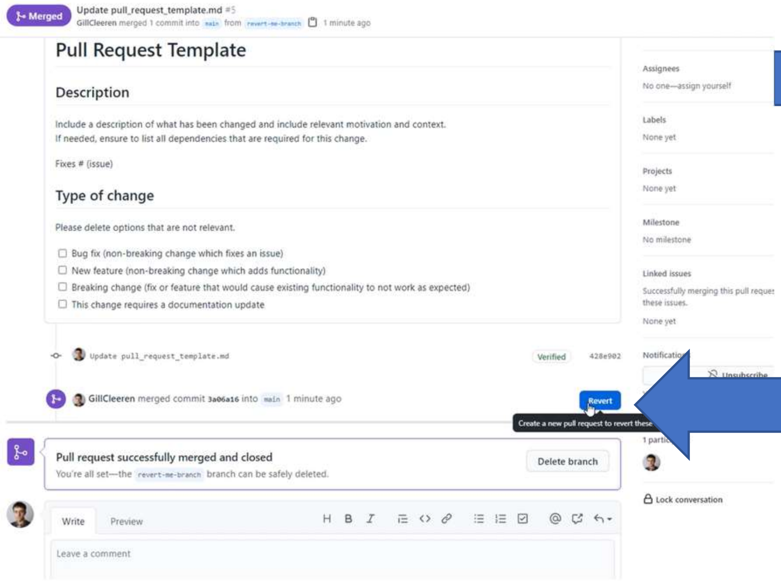


```
# Pull Request Template

## Description

Include a description of what has been changed and include relevant motivation and context.
If needed, ensure to list all dependencies that are required for this change.

Fixes # (issue)

## Type of change

Please delete options that are not relevant.

- [ ] Bug fix (non-breaking change which fixes an issue)
- [ ] New feature (non-breaking change which adds functionality)
- [ ] Breaking change (fix or feature that would cause existing functionality to not work as expected)
- [ ] This change requires a documentation update

```

X

to add:

- Reverting Pull Request



Merged Update pull_request_template.md #5
GillCleeren merged 1 commit into `main` from `revert-me-branch` 1 minute ago

Pull Request Template

Description

Include a description of what has been changed and include relevant motivation and context.
If needed, ensure to list all dependencies that are required for this change.

Fixes # (issue)

Type of change

Please delete options that are not relevant.

- □ Bug fix (non-breaking change which fixes an issue)
- □ New feature (non-breaking change which adds functionality)
- □ Breaking change (fix or feature that would cause existing functionality to not work as expected)
- □ This change requires a documentation update

`Update pull_request_template.md` Verified 428e902

GillCleeren merged commit `3a06a16` into `main` 1 minute ago

Create a new pull request to revert these

Pull request successfully merged and closed
You're all set—the `revert-me-branch` branch can be safely deleted.

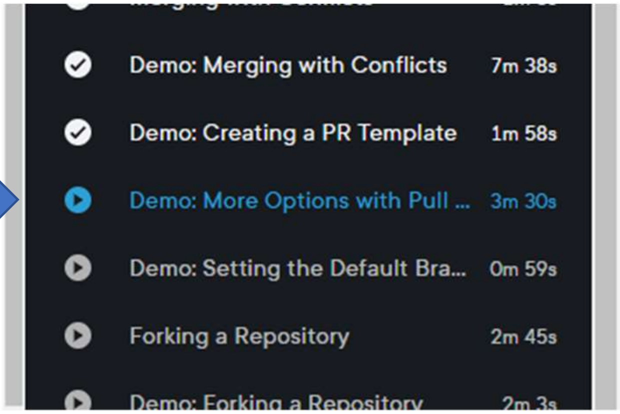
Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

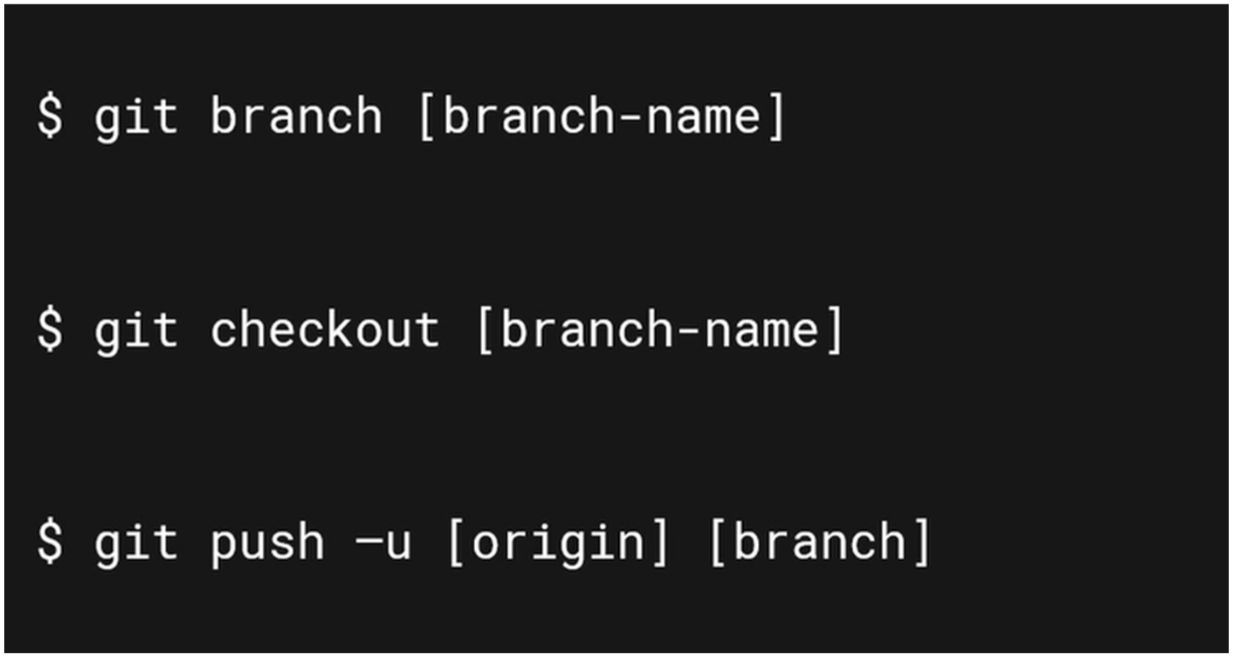
Milestone
No milestone

Linked issues
Successfully merging this pull reques
these issues.
None yet



Demo: Merging with Conflicts	7m 38s
Demo: Creating a PR Template	1m 58s
Demo: More Options with Pull ...	3m 30s
Demo: Setting the Default Bra...	0m 59s
Forking a Repository	2m 45s
Demo: Forking a Repository	2m 3s

X



```
$ git branch [branch-name]

$ git checkout [branch-name]

$ git push –u [origin] [branch]
```

X

Defining a Branching Strategy

Set of rules

Define workflow for branches

Different options exist

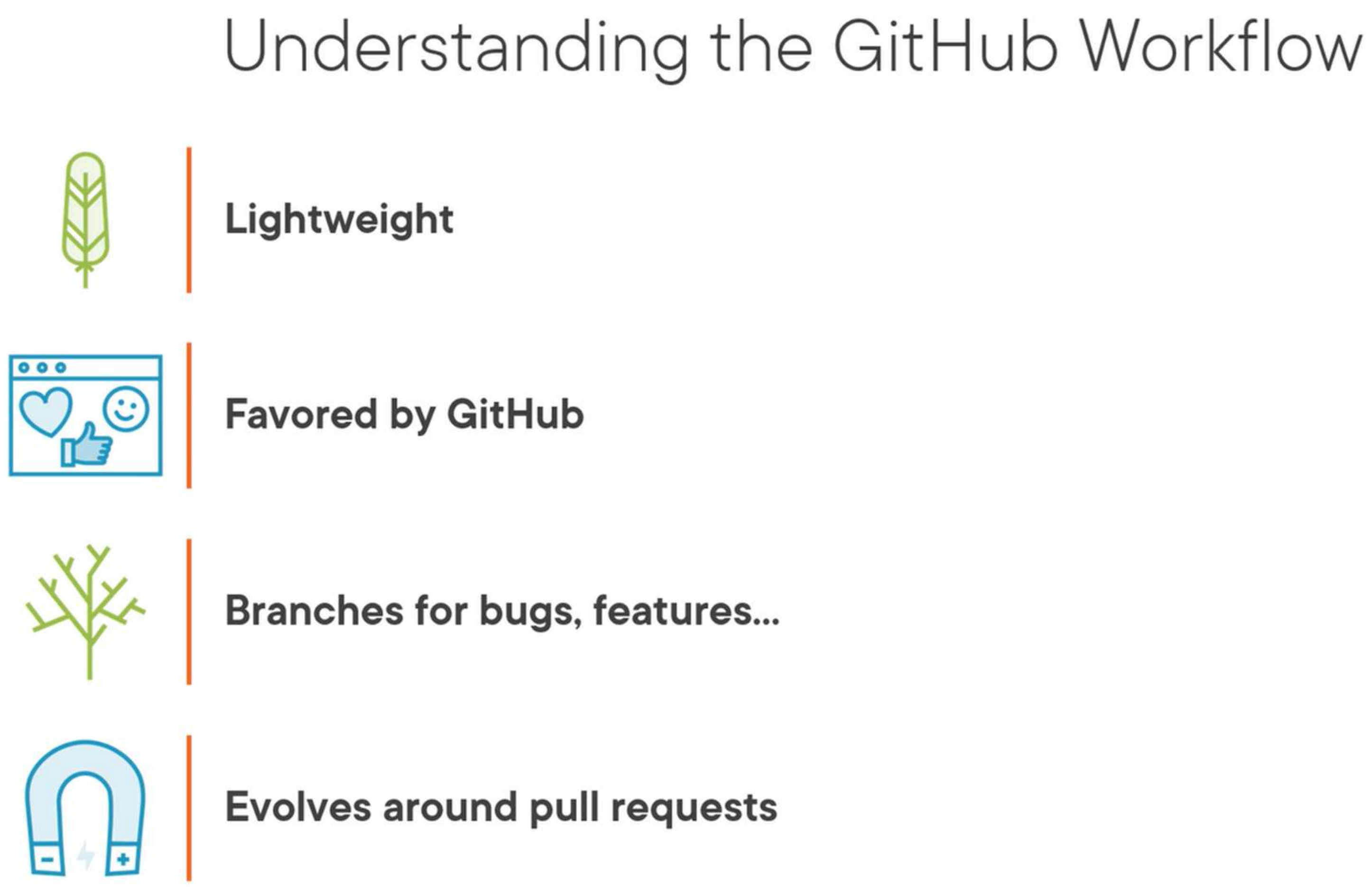
Branching Strategies

Centralized workflow

Gitflow workflow

Forking workflow

GitHub workflow



Understanding the GitHub Workflow

- **Lightweight**
- **Favored by GitHub**
- **Branches for bugs, features...**
- **Evolves around pull requests**

X

Pull Requests

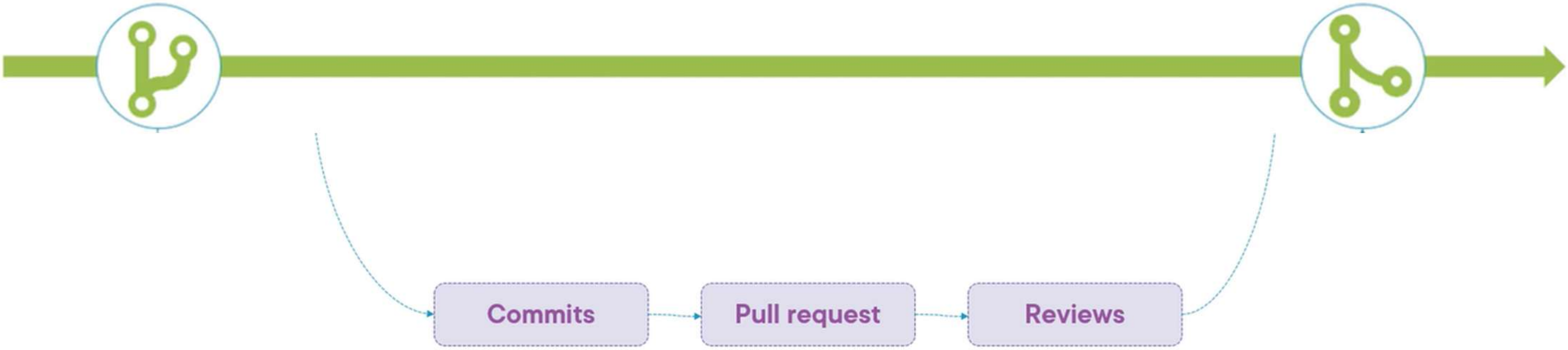
Announce a push to a branch

Discuss
Review changes
Add more commits

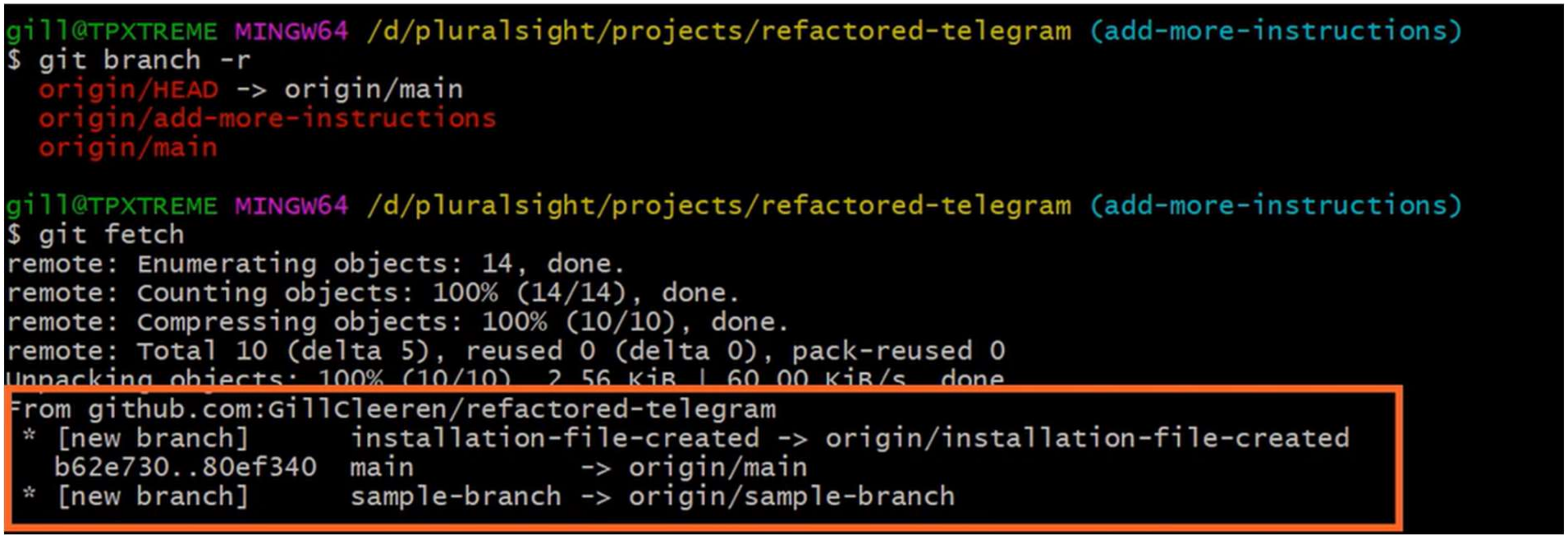
Merge

X

GitHub Flow



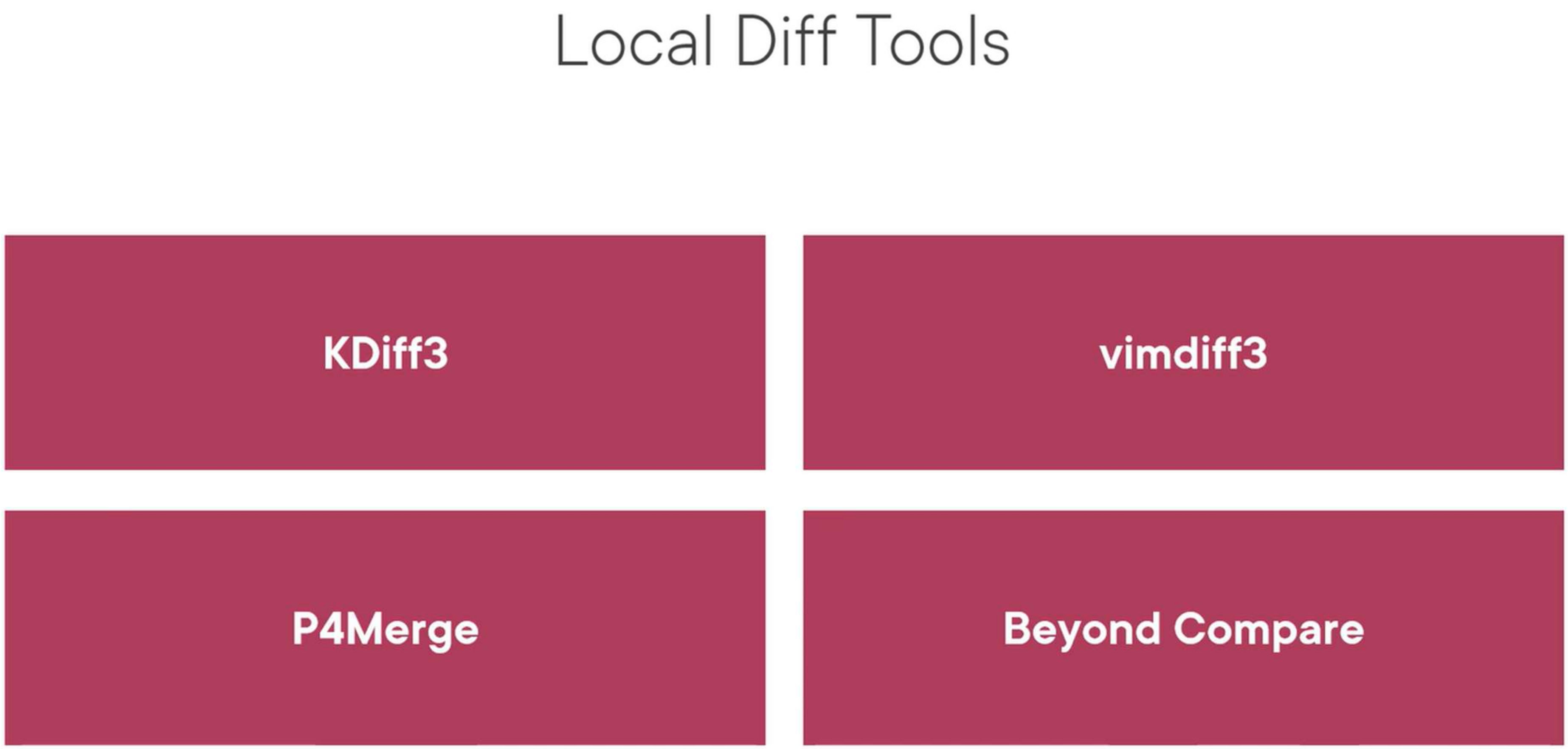
X



```
gill@TPXTREME MINGW64 /d/pluralsight/projects/refactored-telegram (add-more-instructions)
$ git branch -r
  origin/HEAD -> origin/main
  origin/add-more-instructions
  origin/main

gill@TPXTREME MINGW64 /d/pluralsight/projects/refactored-telegram (add-more-instructions)
$ git fetch
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 10 (delta 5), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (10/10), 2.56 KiB | 60.00 KiB/s, done.
From github.com:GillCleeren/refactored-telegram
 * [new branch]      installation-file-created -> origin/installation-file-created
   b62e730..80ef340  main          -> origin/main
 * [new branch]      sample-branch -> origin/sample-branch
```

X

For merge conflicts

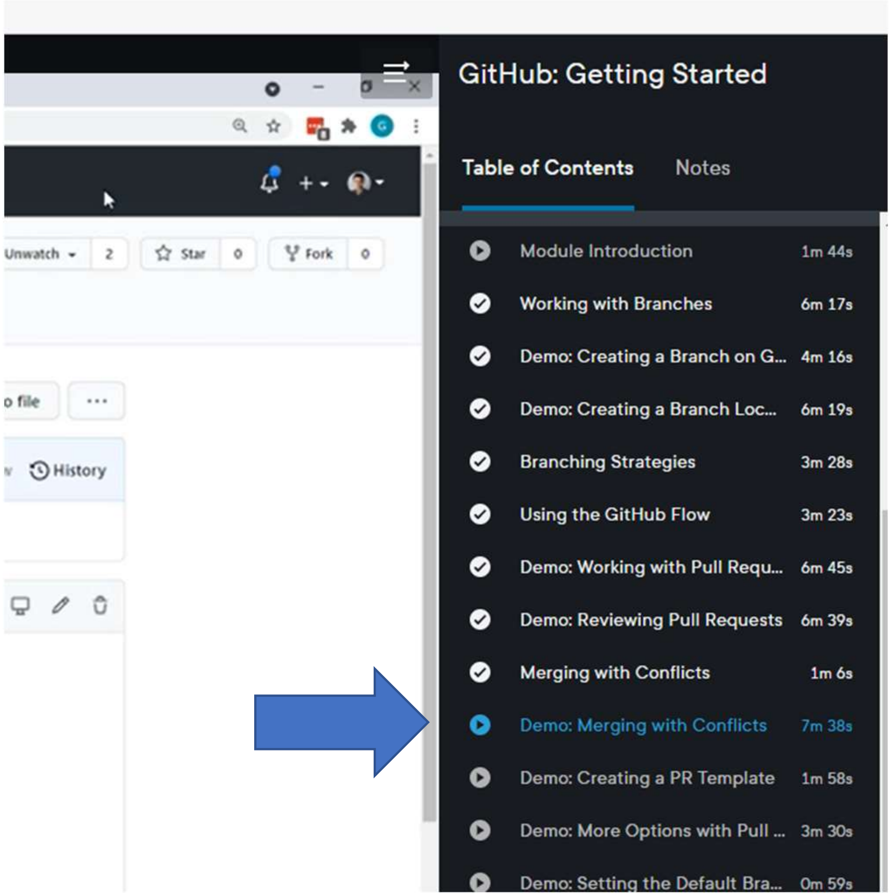


Local Diff Tools

- KDiff3
- vimdiff3
- P4Merge
- Beyond Compare

X

See this vid for merge conflicts: [GitHub: Getting Started | Pluralsight]



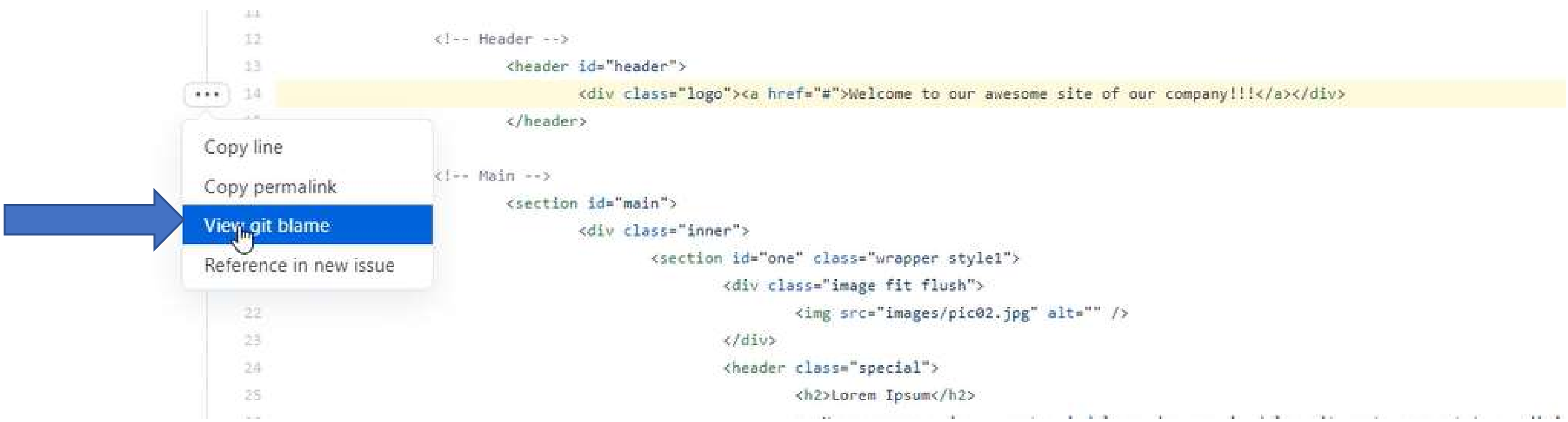
GitHub: Getting Started

Table of Contents   Notes

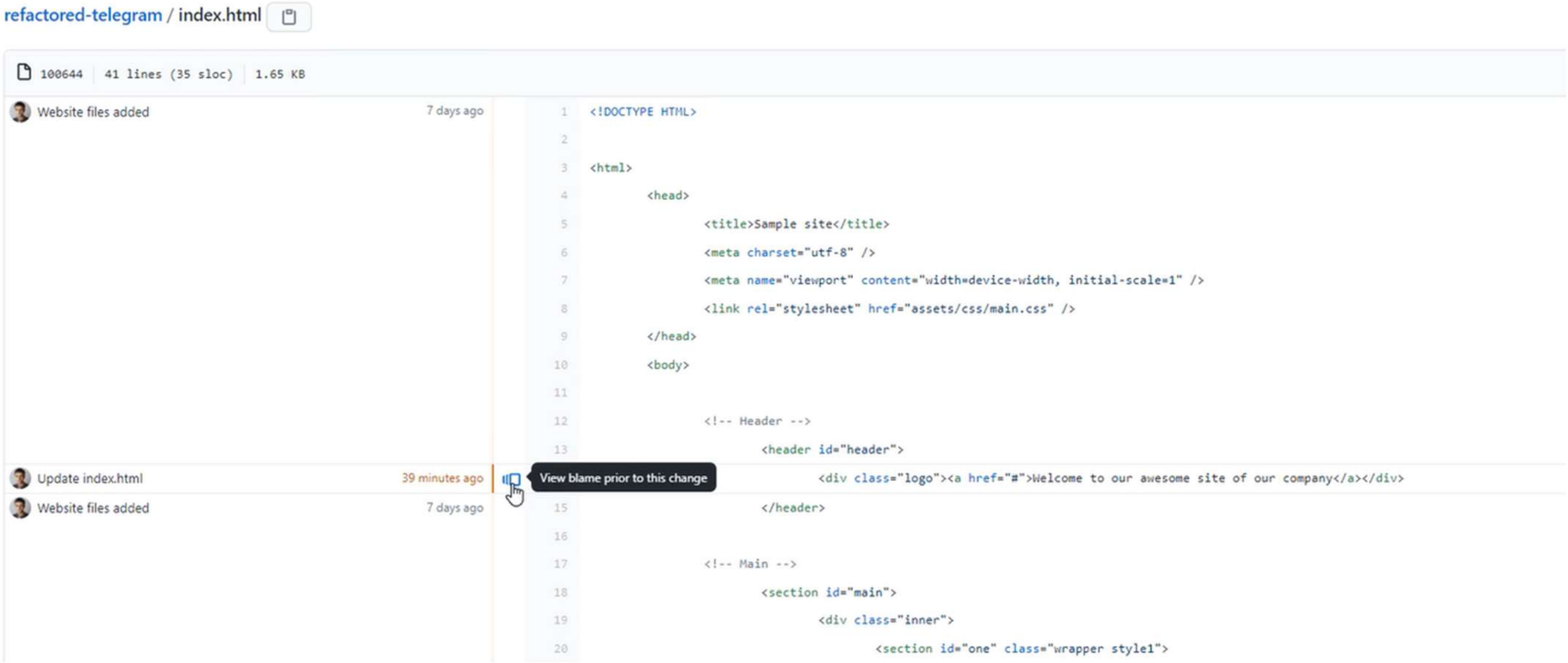
Module Introduction	1m 44s
Working with Branches	6m 17s
Demo: Creating a Branch on G...	4m 16s
Demo: Creating a Branch Loc...	6m 19s
Branching Strategies	3m 28s
Using the GitHub Flow	3m 23s
Demo: Working with Pull Requ...	6m 45s
Demo: Reviewing Pull Requests	6m 39s
Merging with Conflicts	1m 6s
Demo: Merging with Conflicts	7m 38s
Demo: Creating a PR Template	1m 58s
Demo: More Options with Pull ...	3m 30s
Demo: Setting the Default Bra...	0m 59s

X

Shows all the changes made to that line (see next slide)



```
<!-- Header -->
        <header id="header">
                <div class="logo"><a href="#">Welcome to our awesome site of our company!!!</a></div>
        </header>

<!-- Main -->
        <section id="main">
                <div class="inner">
                        <section id="one" class="wrapper style1">
                                <div class="image fit flush">
                                        <img src="images/pic02.jpg" alt="" />
                                </div>
                                <header class="special">
                                        <h2>Lorem Ipsum</h2>
```



refactored-telegram / index.html

100644 | 41 lines (35 sloc) | 1.65 KB

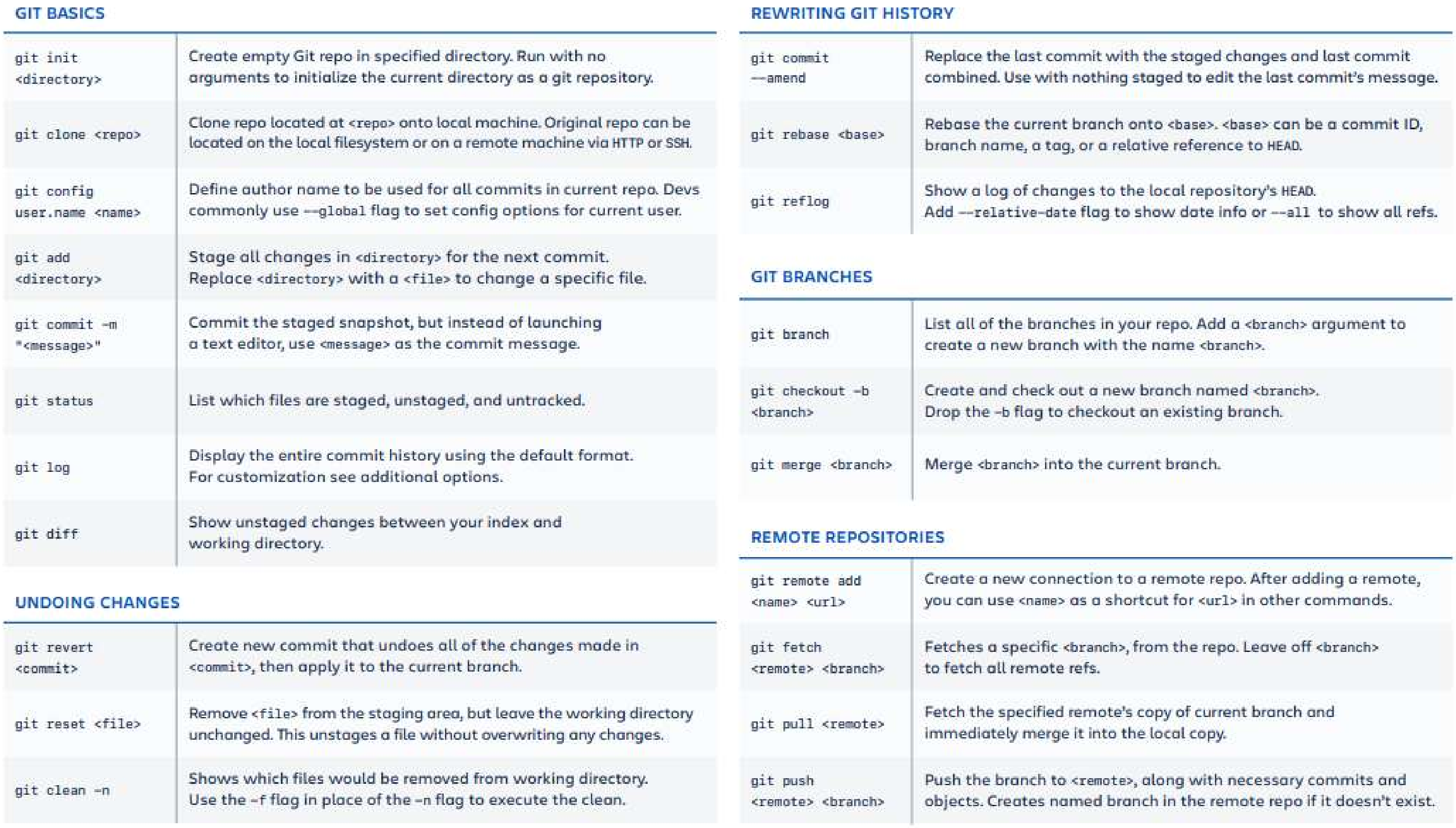
Website files added 7 days ago

Update index.html 39 minutes ago

Website files added 7 days ago

```
<!DOCTYPE HTML>

<html>
        <head>
                <title>Sample site</title>
                <meta charset="utf-8" />
                <meta name="viewport" content="width=device-width, initial-scale=1" />
                <link rel="stylesheet" href="assets/css/main.css" />
        </head>
        <body>

                <!-- Header -->
                        <header id="header">
                                <div class="logo"><a href="#">Welcome to our awesome site of our company</a></div>
                        </header>

                <!-- Main -->
                        <section id="main">
                                <div class="inner">
                                        <section id="one" class="wrapper style1">
```

Summary of key Git Commands



GIT BASICS

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

UNDOING CHANGES

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

REWRITING GIT HISTORY

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

GIT BRANCHES

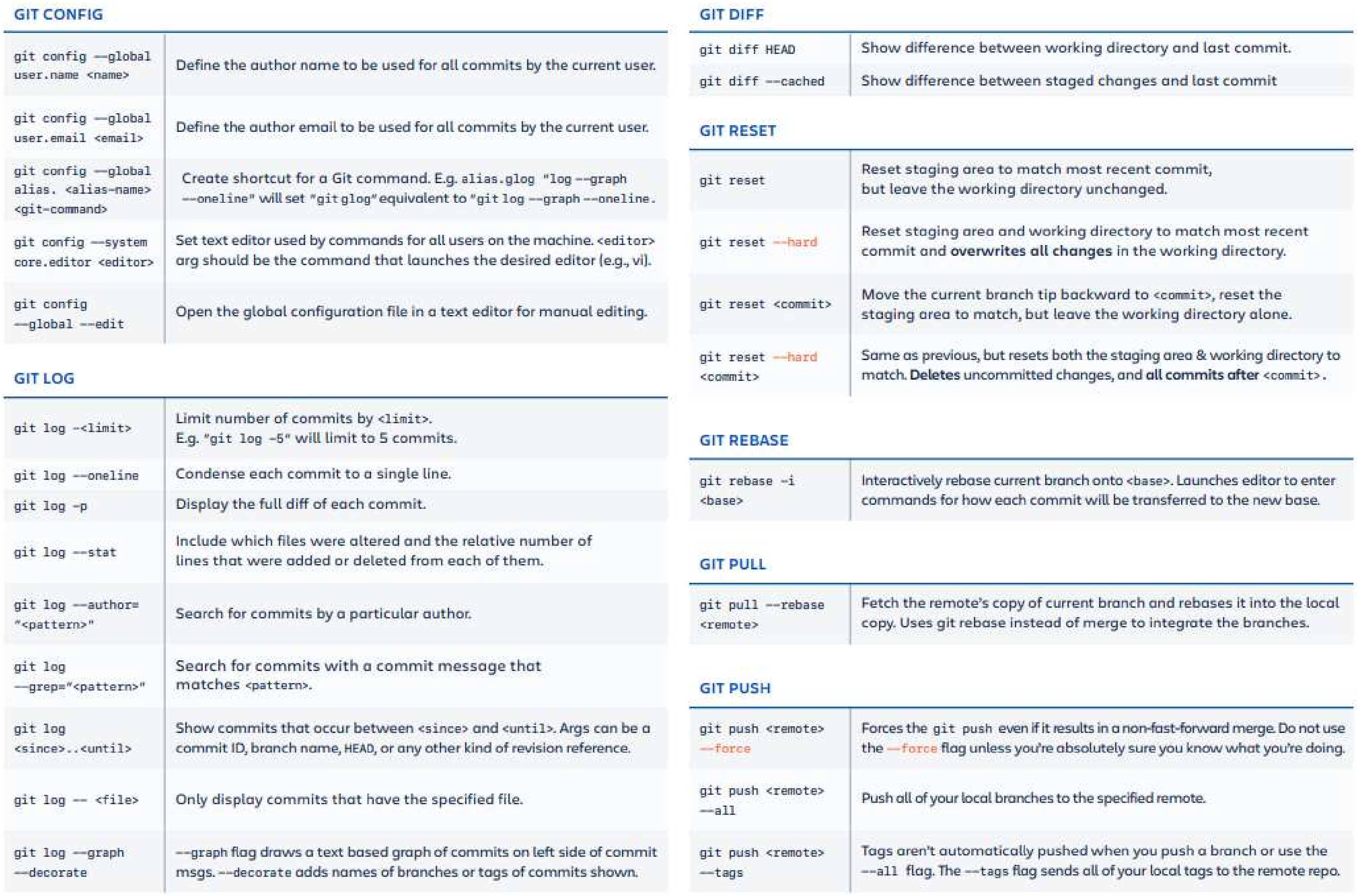
`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the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

REMOTE REPOSITORIES

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

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

Summary of key Git Commands



GIT CONFIG

git config --global user.name <name>	Define the author name to be used for all commits by the current user.
git config --global user.email <email>	Define the author email to be used for all commits by the current user.
git config --global alias. <alias-name> <git-command>	Create shortcut for a Git command. E.g. alias.glog "log --graph --oneline" will set "git glog" equivalent to "git log --graph --oneline.
git config --system core.editor <editor>	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).
git config --global --edit	Open the global configuration file in a text editor for manual editing.

GIT LOG

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

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

GIT RESET

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

GIT REBASE

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

GIT PULL

git pull --rebase <remote>	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

git push <remote> --force	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.
git push <remote> --all	Push all of your local branches to the specified remote.
git push <remote> --tags	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

X

X

X

X

X

X

X

X

X