Git Survival Kit

Please note: the following assumes you already have (1) a basic understanding of Git and GitHub; (2) Git installed on your local machine; (3) a GitHub account. See the Appendix for more info on the basics, including an overview of Git.

There are many ways to collaborate using Git. One simple workflow is called the "ABC Workflow" (a version of feature branching) and will be used in the following sections. In this workflow, we follow the **three rules ("ABC")** and use **three workflow stages ("Create, Push, Pull")**. There are also many ways to execute git commands, including the **GitHub Desktop** and the **Git Bash.** While the **Git Bash** is included with the default install of the git client and will give you a greater understanding of how things work under the hood, **the simple way to interact with Git is by using the GitHub Desktop.**

The Rules

A. **A**lways merge! (never rebase)
B. **B**ranch! (before you do your work)
C. **C**reate pull requests! (on GitHub)

More specifically, do all work on feature branches, not on the main branch. When you want to combine work, always use the "merge" command, never rebase. When ready to add your feature branch into the main branch, create a pull-request on GitHub instead of directly merging into the main branch. Use your poetic license if you know what you're doing.

The Stages

1. **Create** (feature branch + make edits)
2. **Push** (feature branch)
3. **Pull** (request + merge + delete feature branch)

GitHub Desktop

Installing GitHub Desktop

https://docs.github.com/en/desktop/installing-and-authenticating-to-github-desktop/installing-github-desktop,

GitHub Desktop Documentation

https://docs.github.com/en/desktop

Find (or create) repo

We can't contribute to a repo unless we find it, clone it, and set it as our working directory.

1. If cloning an existing repo:
    a. GitHub Desktop > File > Clone Repository > GitHub.com or URL
    b. Make sure local path is where you want to clone
2. If creating your own new repo…
    a. For solo work:
        i. https://github.com/new
    b. In either case:
        i. give new repo a name
        ii. give a description
        iii. **make "Private" (unless you want to share with the whole world!!!)**
        iv. add a README file
        v. add a .gitignore (see Appendix: "Ignoring files with .gitignore")
        vi. click “Create Repository”
        vii. **complete all steps from “1.”**

Stage #1 - Create (feature branch + make edits)

Change directory

You are ready to start working!

1. GitHub Desktop > Current Repository (top left) > click dropdown and select the repo you want

Get latest from main

1. GitHub Desktop will automatically refresh your local repo when you switch repositories and as you make changes.
2. If the remote repo has changes your local repo doesn't have yet, click to pull those down.
    a. Always pull the latest on main before creating a new feature branch off of main

Switch to branch

1. To create a new branch: Current Branch > New branch
    a. Feature branch names should be standard across the team (e.g. start with "feat/" and be short + descriptive + hyphenated like "feat/new-line-graph")
    b. Make sure your new branch is "based" on the branch you want (in most cases, your base should be "main")
2. To switch to an existing branch: Current Branch > click drop down

Making changes and committing

1. GitHub Desktop simplifies your life by treating working directory changes as already "tracked" staging directory changes. This means you can make edits and skip right to creating a commit.
2. To check your historical commits, click "History" under "Current Repository".
    a. Any commits with an upward arrow on the right side indicate un-pushed commits.

Stage #2 - Push (feature branch)

Sync branch with main

You are ready to push your feature branch to the remote repo on GitHub! However, it's possible that in the meantime there were updates to the main branch, and you should pull these into your feature branch.

1. Checkout your feature branch by selecting it as your current branch
2. Branch > Update from main

Push branch

1. Push your feature branch to GitHub by clicking "Push origin"

Stage #3 - Pull (request + merge + delete feature branch)

Create pull request

You've pushed your branch to GitHub and are ready to make a pull request!

1. Navigate to the repo on GitHub > "Pull Requests" > "New pull request"
2. **base: main <-- compare: <your-branch>**
3. "Create Pull Request" > change PR message or comments > "Create Pull Request"

Best practice is to discuss changes with a team member, have them locally pull down the branch in the PR, verify the code works, then approve your PR.

Merge PR

When ready to accept the pull request changes, check if a merge conflict(s) exists.

1. If no, follow the prompts to merge the change into the main branch.
2. If yes, best practice is to update your PR by syncing with the main branch, resolving the conflicts from that sync, then re-pushing the branch. Otherwise, someone looking at your PR will have to do those same steps but without knowing how you intended to resolve the conflicts.
    a. Another option is to keep your PR as-is and resolve the merge conflicts within GitHub via "Resolve conflicts" > fix each file > "Mark as resolved" > "Commit merge".

Delete remote / local branches

After a successful PR merge, you can "Delete branch" on GitHub if you don't want to keep it. Note this only deletes the remote branch. To delete the local branch in GitHub Desktop, right click from the branch drop down > Delete (note you never want to delete “main”).

Pull on main

After a successful pull request, remember to switch back to main and pull down those local changes.

Git Bash

The Git Bash will give you a greater understanding of how Git works and is necessary for advanced troubleshooting techniques that can't be accomplished in the GitHub Desktop. However, it's not recommended to use the Git Bash for important work if you haven't used it previously. There are many obscure commands with precise syntax, and "undoing" a command can be tricky. If you want to learn, start by creating a "dummy" personal repo and just use simple text files.

Find (or create) repo	Same steps as for GitHub Desktop
cd "P:\Documents\Git"	> Make a "Git" parent folder for all your repos (or use the same one if you've cloned other repos before). > Open Git Bash + navigate to directory where you want the local repo to live
git clone {repo_name} <optional-folder-name>	Clone the repo. This action will create a new folder whose name will be either (1) the last portion of the name of cloned repo or (2) the <optional-folder-name> if you specify one.
cd {repo_name}	Change to the directory that was just created

Stage #1 - Create (feature branch + make edits)

Change directory	You are ready to start working! • Change directories to the local repo you want. • Paths with spaces need to be encapsulated with double quotes.
cd <path-to-your-local-repo>	• Change to target directory
Get latest from main	Your local repo does not automatically refresh when the remote repo is updated; it needs to be manually refreshed Always pull the latest on main before creating a new feature branch off of main
git checkout main git pull	Checkout main + pull latest
Switch to branch	Feature branch names should start with "feat/" and be short + descriptive + hyphenated (e.g. feat/new-line-graph)
git checkout -b <new-branch>	**Option #1 - creates <new-branch> from main + checks out <new-branch> + switches you to <new-branch>**
git checkout <existing-branch>	**Option #2 - checkout an existing branch**
Verify current branch	
git branch -a	**Option #1 - return all local and local-remote-tracking branches; your current local branch will have an asterisk.**
git branch	**Option #2 - return local branches; your current local branch will have an asterisk.**
Add changes to tracking	Make some edits and add them to tracking when ready.
git add <file-name>	**Option #1 - add specific untracked file (i.e. working directory) to tracking (i.e. staging area)**
git add *.<extension-name>	**Option #2 - add currently untracked files with extension <extension-name> (e.g. *.txt) to tracking**

git add .	**Option #3 - add all currently untracked files (except those in your local and global .gitignore files)**
Check status	Read the messages! Untracked files will show in red, and tracked files will show in green.
git status	
Commit files	Commit tracked (i.e. staging area) files to local repo Untracked files (i.e. working directory) are not included.
git commit -m "<commit-message>"	
Check log	If the log is greater than what the console shows, there will be a colon (":") at the end • To continue browsing the log, scroll using the "down" arrow key • **To quit the log, type "q"**
git log	**Option #1 - generic log**
git log --oneline	**Option #2 - log with one line per commit**
git log --oneline --stat	**Option #3 - log with one line per commit, additional lines for each file affected in the commit**

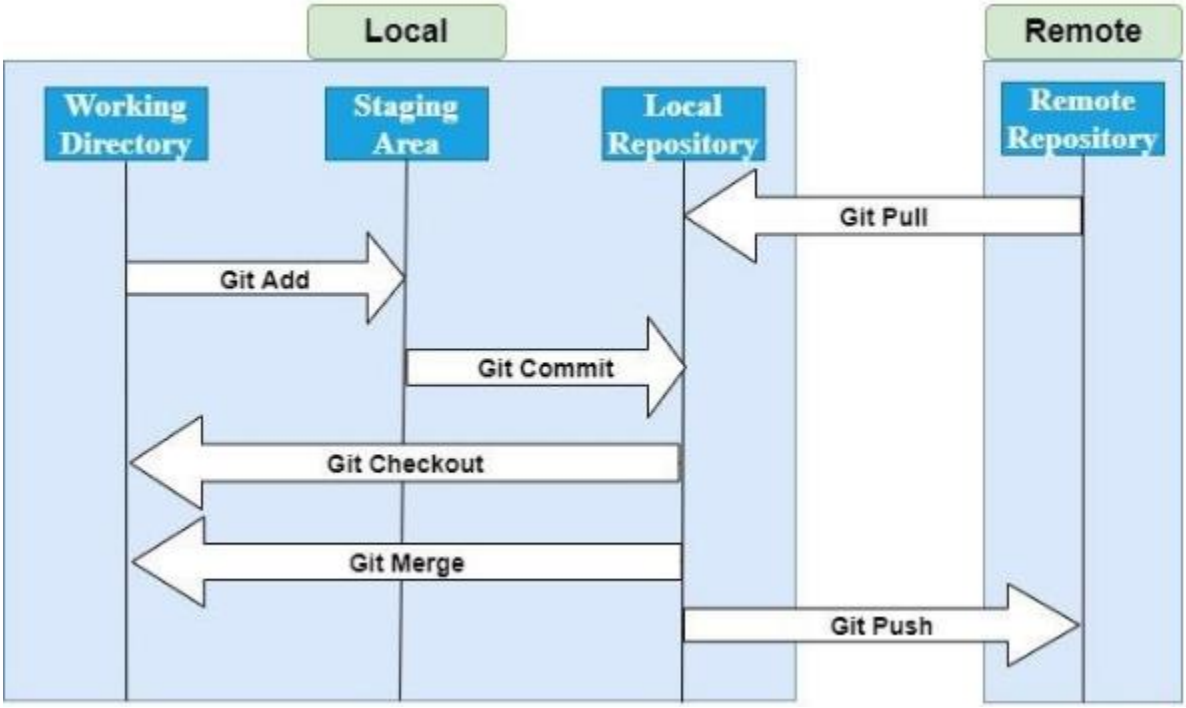
Stage #2 - Push (feature branch)

Sync branch with main	You are ready to push your feature branch to the remote repo on GitHub! • Sync your branch with main before making a PR • It's also good to sync with main on a regular basis (e.g. once a day while working in a repo)
git checkout <my-branch> git pull origin main	**Option #1 - sync branch by pulling latest changes from the remote main + merging into your current branch**
git checkout main git pull git checkout <my-branch> git merge main	**Option #2 - same as above but more atomic: explicitly getting latest for main + merging into your current branch**
Push branch	Actually push your branch to GitHub
git push origin <my-branch>	Note you can push your branch to the remote repo any time, regardless of whether it's going to be in a future pull request.

Stage #3 - Pull (request + merge + delete feature branch)

Create pull request	Same as GitHub Desktop
Merge PR	Same as GitHub Desktop
Delete remote / local branches (optional)	After a successful merge, you can "Delete branch" on GitHub if you don't want to keep it • Note this option on GitHub only deletes the remote branch • Note that every branch pushed to GitHub has 3 related but separate types of branches: 1. Local branch 2. Local remote-tracking branch 3. Remote branch
delete branch on GitHub git checkout main git pull git fetch origin --prune git branch -d <my-branch> git branch -a	To delete the local and local remote-tracking branches after merging + deleting the remote branch, follow these steps. You no longer need the local branch after a successful merge, but **be careful when deleting local branches!**

Appendix



This is how Git works:

A. You **make** some changes; these are **untracked** in your **Working Directory**
B. You **add** those changes; these are now **tracked** in your **Staging Area** (also called the **Index**)
C. You **commit tracked** changes; these are now in your **Local Repo**
D. You **push commits**; these are now in the **Remote Repo** (in our case, GitHub)

Installing Git

The Git client can be found here: https://www.git-scm.com/

Ignoring files with .gitignore

Some files you **never** want to include in GitHub

- **Passwords**
- **Private info**

Some files you **could** put into GitHub **but generally shouldn't**

- **Anything that can be generated (like R plots)**
- **User-specific config files**
- **Large "binary" (aka "non-text") files like...**
  - **.pdf**
  - **.xlsx**
  - **.docx**
  - **.jpeg**

A .gitignore file helps exclude files from being added to your Git commits. This file lives in your repo and helps you track the specific changes you want to add to source control. You can ignore specific files, extensions, directories, or a combination of all three. Ignored files can still be edited, but these edits will not be picked up by git. Note that ".gitignore" is the full name of this file and not just an extension.

1. Sample contents of a .gitignore is below.

```
# ignore all jpeg and pdf files
*.jpeg
*.pdf
```

2. Another .gitignore strategy is to ignore *everything*, then "un-ignore" certain files or extensions. This gives you more explicit control but requires more micro-managing.

```
# ignore everything...
*.*

# ...except these extensions, which will not be ignored...
!.gitignore
!*.txt
!*.sql
!*.R
!*.rdl

# ...but then re-ignore these specific files
*logHistory*.txt
```