Git Cheat Sheet

Updated 2/17/2022

Definitions and Nomenclature

- **Git:** A tool for version control and change-tracking of software.
- **Command Line:** Software that allows you to programmatically interface with a Git repository. The specific software may depend on your operating system.
- **Working directory:** Where files are edited, added and discarded.
- **Repository**: A collection of files that are managed using Git.
- **Local**: The copy of the repository that exists on your computer.
- **Remote**: The “reference” copy of your repository that exists on Bitbucket, Github, etc. The local repository is compared against the remote repository to track changes.
- **Main**: The "official” version of the repository. This represents the core code of the repository.
- **Branch:** An alternate version of the repository that “branches off” from the main. Branches may contain features that are in development, but are not yet ready to be incorporated into the official release. In a tree analogy, the main is the trunk, and branches are, well, branches.

Note: A word in brackets (like `<this>`) indicates a placeholder for a name of a directory, file, branch, etc. Brackets do not need to be included when the actual line of code is run.

Getting started - command line directory navigation

Git command	What it does
`cd <path-to-directory>`	Changes the current working directory in the command line to the specified directory.
`cd`	Changes the current working directory to the root directory of your machine.

Note: To include spaces in the directory name, the space characters must be escaped using a backslash. For example, to navigate to the directory “Path with Spaces”: `cd Path\ with\ spaces/`

Getting started - creating a new repository

First, create the remote repository (this is where your files will eventually live).

In GitHub:
1. Navigate to your repositories
2. Click the new repository button
3. Name the repository and add a description
4. Select privacy: typically private, unless you want to share with anyone on the internet
5. Check yes to add README.md
6. Select yes to add .gitignore and select an appropriate template, if desired
7. Click create repository

In Bitbucket:
1. Click the + icon on the left sidebar, then click “repository”
2. Select the project this repository is related to. You may create new projects.
3. Name the repository and select privacy.
4. Name the default branch (e.g., ‘main’)
5. Ensure that both README and .gitignore are included
6. Click create.

We now have an empty remote repository that files can be added to. We will connect this repository to files on our computer via “cloning”.

Cloning a repository

Cloning is creating a local version of a remote repository that already exists. If a repository is empty (like a newly created repository), cloning is the next step in connecting the remote repository to your local machine. If a repository is full of files, cloning creates a local version of those files on your machine.

In the hosting service of your choice:
In GitHub
1. Click the "< > code" button
2. Under "clone", copy the https link to your clipboard

In Bitbucket
1. Navigate to your repository. Click the "source" tab in the repository sidebar.
2. Click the "clone" button in the upper right-hand corner. Copy the line of code shown () to your clipboard.

Then:
In the command line
1. Navigate to the directory on your computer where you want the local repository to live.
2. Paste the line of code from GitHub/Bitbucket and run. This will create a folder with the same name as your repository in the directory you chose. Files in the remote repository should be visible there.

If you are creating a new repository and have files you would like to populate that repository with, you can now move your files into this directory!

Making changes

When using git to track changes, we must consider three places where files can live along the process of "saving our progress" when editing these files.

1. The working directory - where edited files live before any progress is saved at the repository-level (files themselves are saved just like any other file in Word, MATLAB, etc).
2. The local repository - where changes to files are saved locally using git.
3. The remote repository - where changes to files are saved in the cloud using git.

The following commands are useful for making changes to files in a repository:

Git command	What it does
`git status`	Reports the current state of the repository. Lists modified files, untracked files, and whether the local repository is up-to-date with the remote.
`git fetch`	Checks for new updates in the remote repository. These changes will be reported by `git status` but will not overwrite the local repository.
`git pull`	Checks for updates in the remote repository and pulls these changes into the local repository. Note that this may produce errors if both the remote repository and the local repository contain separate changes to the same file(s) (see "Merging" below).
`git add <file or directory name>`	Stages the requested file for committing.
`git add -A`	Stages all untracked and modified files for committing.
`git restore <file or directory name>`	Resets the requested file to the version recorded in the remote repository. Essentially, this discards changes in the local repository.
`git commit -m '<message text>'`	Commits all staged files, and logs the commit with the input message. Essentially, this is saving local changes and preparing them to be pushed to the remote (they have not yet been saved at the remote level).
`git push`	Pushes committed changes to the remote repository. After this command, committed changes will be reflected in the repository. Note: in order to push changes, your local repository must be up-to-date with the current remote (see `git pull`).
`git reset --hard origin <branch name>`	Resets the requested branch of the local repository such that it matches the remote repository's version of that branch. All local changes that do not match the remote are deleted.

A general workflow is described below:

1. Use `git pull` to check for and incorporate any changes from the remote repository. Resolve merge conflicts (see “Merging” below).
2. Edit local repository files or add/remove files from the repository. Use `git status` to see these changes.
3. Use `git add` to stage files with changes that you would like to commit to the remote repository.
4. Use `git commit -m` to commit these changes. Type in a message that describes the changes.
5. Use `git push` to push the local committed changes to the remote repository.

Merging

Merging is necessary when the local and remote versions of a repository have diverged, each containing different committed changes that are not reflected in the other. This often happens when 2 or more people are working with the same repository, or if a repository is managed by one person on two separate computers. Merging essentially reconnects these two parallel versions of the repository by combining the committed changes present in each.

A merge that unites the diverged local and remote repositories occurs when `git pull` is run under certain criteria. Typically, there are 3 outcomes:

Failure to merge

A merge will fail when the working directory contains uncommitted changes that would be overwritten by committed changes in the remote. To resolve: either commit or discard changes in the working directory and pull again.

Automatic Merging

Automatic merging occurs when the separate changes in the local and remote repositories do not conflict and can be combined. In this case, the command line will ask you to type a commit message for the merge, and then the merge will be completed.

The default text editor (at least for Windows) is a text editor called vim, which can be pretty confusing. Follow these steps to type your message after the prompt appears:

1. Press “i” (i for insert)
2. Type a merge message for why this merge is occurring.
3. Press “esc” (escape) to exit the insert.
4. Type “:wq” (write and quit) and press enter

This will save your merge message. Run `git commit` to complete the merge locally, and `git push` to push the merge to the remote.

Merge Conflicts

Merge conflicts occur when the committed changes in the remote repository are incompatible with the committed changes in the local repositories and cannot be combined or otherwise reconciled automatically. These conflicts must be manually resolved.

To resolve: open each file listed as “both modified”. Manually select changes to include in the final version, and use `git add <file or directory name>` to stage the final version of the file. Finally, use `git commit` to commit the final versions to the locally repository, and `git push` to push the merge to the remote.

Branches

Branches are ways to create parallel versions of the core code of a repository. They are useful for developing experimental features or large changes to your code without disrupting use of the official release (the main branch), while retaining the ability to track changes.

Git command	What it does
`git branch <branch_name>`	Creates a local branch named `<branch_name>`, which branches off of whatever the current branch. `<branch_name>` is a copy of the working directory at the time of branching (and includes all unstaged files in the working directory).
`git checkout <branch_name>`	Switches the current branch to be `<branch_name>`. Note that a branch cannot be checked out if there are uncommitted changes in the working directory that would be overwritten by the checkout.
`git push -u origin <branch_name>`	Pushes the branch "branch_name" to the remote repository so that local changes can be tracked against it as a reference. This command should be run for the initial push, but afterwards, changes can be sent to the remote repository using `git push` as normal.
`git branch -a`	Lists all branches and all remotes that are on the local machine.
`git branch -vv`	Lists all branches on the local machine, and the remote that they are tracking.
`git merge <other_branch>`	Merges `<other_branch>` into the *current* branch. See "Merging" above for guidelines on how to manage the merge.
`git push origin -- delete <branch_name>`	Deletes the branch `<branch_name>` on the remote repository.
`git push origin -d <branch_name>`	Deletes the branch `<branch_name>` on the local repository. This will not delete this branch on other machines.

A typical workflow might look like:

1. Begin to develop a new feature in the main branch.
2. Realize that these changes need more development and should be in their own branch.
3. Use `git branch new_feature` to make a new branch named `new_feature` that is a copy of main. Unstaged changes are carried over.
4. Use `git checkout new_feature` to switch your current branch to the `new_feature`.

5. Stage and commit any new unstaged and edited files, and push the initial commit using `git push -u origin new_feature`.
6. Continue to work on the new feature in branch `new_feature` by pulling, adding, committing, and pushing as normal while on this branch.
7. Keep `new_feature` up to date with any changes to the main branch by periodically using `git merge main` to merge the main branch into `new_feature`
8. When the feature is ready to be implemented in a future release, merge into main by either
    a. Checking out the main branch via `git checkout main` and use `git merge new_feature` to merge `new_feature` into main
    b. Submit a pull request on GitHubBitBucket

Branches can also be created using BitBucket by doing the following:

1. Click "Branches" on the sidebar. In the top-right corner, click "Create branch".
2. Choose a name for your branch. If desired, select the type of branch (i.e, hotfix), which will add a prefix to your branch.
3. To use this branch on your local machine, run `git fetch && git checkout <branch_name>`. This will add the branch to your local repository and switch to that branch. The new branch will automatically track its remote counterpart.
4. Verify branch creation by running `git branch -a`. Verify branch tracking by running `git branch -vv`. If you need to switch back to the main repository, you can do so by running `git checkout main.`