UNIX

Bash profile

When you launch Git Bash, it will refer to a file located in the home directory (usually `/c/Users/<username>`). The file may be:

- `.bash_profile`
- `.bash_rc`
- Others that are hidden.

Within your profile you will specify aliases, variables, and other features that you want Git Bash to run or have access to every time you run it. That file is not included in this repository.

Command help

- `<command> --help` to get the manual/help page for any command (`man <command>` in UNIX).
- `which <command>` to show you path to the command. If command not found, it is not in the Windows Path variable.

- `<command>` by itself often leads to some helpful information as well.
- CTRL + C at any time to cancel the running operation.

Location

- `pwd` prints working directory.
- `/` shorthand for the root directory.
- `~` shorthand for home directory.
- `cd <directory>` changes directory.
  - `cd -` takes you back to whatever location you were just in.
- `.` is working directory.
- `..` is parent directory.
- `ls` list contents of working directory.
- `ls -alrt` lists contents of working directory, including hidden files, with details in reverse chronological order.
  - `-a` shows hidden files.
  - `-l` shows all details.
  - `-r` reverses order (normally sorts alphabetically).
  - `-t` sorts by timestamp.

Making, Moving, Removing Files

- `mkdir <directory name>` creates a directory.
- `touch <filename.extension>` creates a file.
- `rm <file>` removes a file.
- `rm -fr <directory name>` removes a non-empty directory.
  - `-r` means recursive.
  - `-f` means force in case there higher permissions required.
  - `rm -fr .git` removes git repository tracking.
- `mv <file or directory> <destination path>` moves a file or directory.
- `mv <old file or directory name> <new file or directory name>` renames a file or directory.
- `mv --force <old name> <new name>` renames a file or directory changing capitalization; failing to use the force argument means changes in capitalization will not be picked up by all systems, including Github.
- `cp <filename> <newfilename>` copies a file.

Strings

- `echo "<text here>"` prints text.
- `-` is wildcard meaning any number of any characters. e.g., `ls -.html` will list all files that end with `.html`.
- `?` is a wildcard meaning any character. e.g., `ls file-???.html` will list all files that have file name that follow that format.
- `\` is escape character and is necessary for specifying many symbols and spaces.
- `cut -c1-4` to slice out characters 1:4

Piping

- `|` pipes output from one command to the next command.
- `>` redirects output to a file, overwriting the file.
    - `<` redirects in the other direction.
- `>>` redirects output to a file, appending to the end of that file.
    - `<<` redirects in the other direction. e.g., `echo "<text here>" >> <file name>` appends text to a file.

Looping

- `for i in list; do <command>; done` e.g., `for file in -; do echo "$file"; done` replicates what `ls` does.
- `for file in -\ -; do mv "$file" "${file// /_}"; done` renames all files in current directory by replacing spaces with underscores.

Variables and Functions

- `env` to see all environmental variables.
- `export <variable> <new value>` to modify environmental variables in the session. e.g., `export HOME=/c/Users/D/Desktop` changes `HOME` variable so that `~` refers to the desktop.
- `alias <name>='<code block>'` to create an alias (less robust than variables in terms of where they can be called).
- `<custom variable>='<code block>'` to create a custom variable.

Launching programs

- `start <filename>` to open files with default program.
- `start <program> <filename>` to open files with alternative program- e.g., `start chrome file.md` to open markdown with chrome instead of VS Code

Security and Process Observation

- `who` to see userids of people in the system.
- `/dev/null/` to ignore a command (I suppose you can use this to shut down outputs coming from users in your system?).
- `ps -aef` to get information about running processes.

Some common programs

- `7z`
- `chrome`
- `code`
- `conda`
- `excel`
- `explorer`
- `firefox`
- `git`
- `jupyter lab`
- `jupyter notebook`

- `powerpnt`
- `R.exe` (use --save, --no-save, or --vanilla tag)
- `Rscript`
- `Rstudio`
- `SQLiteStudio`
- `ssh`
- `winword`
- `winpty python` (launching python by itself hangs unless you've assigned an alias)

Other commands

- `ln` to create symbolic links.
- `tar` to create/extract archives.
  - `7z` for using 7zip instead.
- `ssh` for setting up connections with other computers.
- `grep` for pattern searching.
- `sed -e` to use regex for finding/replacing characters
  - `awk` can also be used for string searching.
- `cat` will concatenate all files' contents.
  - `sort` and `-nr` will concatenate all files' contents then alpha or numeric sort them.
- `paste` to combine files into a new file.
- `uniq` and `-c` to give unique values and their counts.
- `head` gives first 10 lines of a file. Can be modified for any number of lines.
  - `tail` gives last 10 lines.
- `less` opens the 'less' viewer which allows you to view files line by line; press 'q' to quit.
  - `more` may do the same thing.
- `wc` gives wordcounts in a file.

Git and GitHub for Version Control

Configuration and Status

- `git config --local -e` to open configuration file for local environment.
- `git config --list` to see current configurations.
- `git config --global <old value> <new value>` to change values. - `git config --system core.longpaths true` to allow pushing/pulling long path names.
- `git status` tells you status of working directory vs staging area and local repository.
- `git log <file name>` gives you a summary of changes made to a file.

Version Control Flow

Initializing

- `git init` to initialize a github repository in the working directory.
- `git clone <URL>` to clone a remote repository to the working directory.

Branching

- `git show-branch --list` to see which branches exist.
- `git branch <branchname>` make new branch.
- `git checkout <branchname>` move to new branch.
    - `git checkout -b <branchname>` to make new branch and move to it in one step.

Staging and Committing to Local Repository

- `git add <filename>` to put file in staging area.
    - `git add .` to put all files in the working directory into the staging area. Files specified in any `.gitignore` files will be ignored.
- `git commit -m '<message>'` to commit staging area to local repository with a message.
    - `git commit -am <message>'` can stage and commit to local repository in one step.
- `git reset --hard <commit hash>` Revert a branch backwards to an earlier commit.

Sending to Remote Repository

- `git remote add origin <url>` to specify remote location for the first time.
- `git push --set-upstream origin <branch>` to send changes to remote repository. You may need to provide user name and password.
    - `git push -u origin <branch>` may do the same with less typing.
    - `git push origin HEAD` will work thereafter.

Pulling from Remote Repository

- `git clone <url>` to clone a repo in the working directory after `git init`.
- `git fetch origin` to synchronize local repository branch based on remote repository branch.
- `git merge <branchname>` to synchronize the working directory with the local repository branch.
    - `git pull origin` does `git fetch origin` and `git merge <branchname>` in one step.
- `git remote` with arguments for viewing and modifying connections.

Synchronizing Different Branches

- `git merge <branchname>` to merge branches, although this will usually be done from GitHub. Some functions for saving changes that are not yet committed before you switch branches:
    - `git stash`
        - `git stash apply`
    - `git pop`

Pruning Dead Branches

- `git remote prune origin` deletes references in local repository to local branches that have been deleted from remote repository.
- `git branch -D <branch>` deletes the branch locally.

Troubleshooting Connection Issues

If you are timing out

- `ssh -T git@github.com` to see if you are timing out. If yes, then the firewall settings may be preventing your connection.

    - `git config --local -e` to open your configuration file. Modify the `url` entry by removing `git@github.com:` or `ssh://git@github.com/` and replacing it with `https://github.com/`. Now your attempts to `pull` should work.

Setting up to push to a repo via SSH if not logged into Github with the owner's account

- `ls -al /c/Users/<username>/.ssh` to check if you have SSH keys.
    - `id_rsa.pub` is likely the one you have used.
- `ssh-keygen -t rsa -b 4096 -C "youremail@domain.com"` to generate a new SSH key if needed. Create your PW (you will not see text entered). - `ssh-keygen -p -f /c/Users/<username>/.ssh/<key file>` to change the password later.
    - `ssh-add /c/Users/<username>/.ssh/<key file>` to add the SSH to the 'ssh-agent'. Login at Github as the repo owner. Navigate to your personal settings, then to SSH and GPG keys.
    - `cat /c/Users/<username>/.ssh/<key file>` to get your public key. Copy all of it, except for the final bit that specifies your email address (include the ssh-rsa at the beginning and any equal signs at the end.) Paste the copied text into Github.
- `git remote set-url origin git@github.com:<username>/<repository>.git` to push to the remote repository using your SSH credentials. You will be asked for your password.