Beginner's Python Cheat Sheet - Git

Version Control

Version control software allows you to take snapshots of a project whenever it's in a working state. If your project stops working, you can roll back to the most recent working version of the project.

Version control is important because it frees you to try new ideas with your code, without worrying that you'll break your overall project. A distributed version control system like Git is also really useful in working collaboratively with other developers.

Installing Git

You can find an installer for your system at git-scm.com/. Before doing that, check if Git is already on your system:

```
$ git --version
git version 2.30.1 (Apple Git-130)
```

Configuring Git

You can configure Git so some of its features are easier to use. The editor setting controls which editor Git will open when it needs you to enter text.

See all global settings

```
$ git config --list
```

Set username

```
$ git config --global user.name "eric"
```

Set email

```
$ git config --global user.email
    "eric@example.com"
```

Set editor

```
$ git config --global core.editor "nano"
```

Ignoring files

To ignore files make a file called ".gitignore", with a leading dot and no extension. Then list the directories and files you want to ignore.

Ignore directories

```
__pycache__/
my_venv/
```

Ignoring files (cont.)

Ignore specific files

```
.DS_Store
secret_key.txt
```

Ignore files with specific extensions

```
*.pyc
```

Initializing a repository

All the files Git uses to manage the repository are located in the hidden directory .git. Don't delete that directory, or you'll lose your project's history.

Initialize a repository

```
$ git init
Initialized empty Git repository in
    my_project/.git/
```

Checking the status

It's important to check the status of your project often, even before the first commit. This will tell you which files Git is planning to track.

Check status

```
$ git status
On branch main
No commits yet
Untracked files:
    .gitignore
    hello.py
    ...
```

Adding files

You'll need to add the files you want Git to keep track of.

Add all files not in .gitignore

```
$ git add .
```

Add a single file

```
$ git add hello.py
```

Making a commit

When making a commit, the `-am` flag commits all files that have been added, and records a commit message. (It's a good idea to check the status before making each commit.)

Make a commit with a message

```
$ git commit -am "Started project, everything
    works."
2 files changed, 7 insertions(+)
create mode 100644 .gitignore
create mode 100644 hello.py
```

Checking the log

Git logs all the commits you've made. Checking the log is helpful for understanding the history of your project.

Check log in default format

```
$ git log
commit dc2ebd6... (HEAD -> main)
Author: Eric Matthes <eric@example.com>
Date:   Feb 27 11:27:07 2023 -0900
    Greets user.
commit bf55851...
...
```

Check log in simpler format

```
$ git log --oneline
dc2ebd6 (HEAD -> main) Greets uer.
bf55851 Started project, everything works.
```

Exploring history

You can explore a project's history by visiting specific commit hashes, or by referencing the project's HEAD. HEAD refers to the most recent commit of the current branch.

Visit a specific commit

```
$ git checkout b9aedbb
```

Return to most recent commit of main branch

```
$ git checkout main
```

Visit the previous commit

```
$ git checkout HEAD^
```

Visit an earlier commit

```
$ git checkout HEAD^^^
```

Visit the previous commit

```
$ git checkout HEAD~1
```

Vist an earlier commit

```
$ git checkout HEAD~3
```

Learning more

You can learn more about using Git with the command `git help`. You can also go to Stack Overflow and search for git, and then sort the questions by number of votes.

Branching

When the work you're about to do will involve multiple commits, you can create a branch where you'll do this work. The changes you make will be kept away from your main branch until you choose to merge them. It's common to delete a branch after merging back to the main branch.

Branches can also be used to maintain independent releases of a project.

Make a new branch and switch to it

```
$ git checkout -b new_branch_name
Switched to a new branch 'new_branch_name'
```

See all branches

```
$ git branch
main
* new_branch_name
```

Switch to a different branch

```
$ git checkout main
Switched to branch 'main'
```

Merge changes

```
$ git merge new_branch_name
Updating b9aedbb..5e5130a
Fast-forward
 hello.py | 5 +++++
 1 file changed, 5 insertions(+)
```

Delete a branch

```
$ git branch -D new_branch_name
Deleted branch new_branch_name
    (was 5e5130a).
```

Move last commit to new branch

```
$ git branch new_branch_name
$ git reset --hard HEAD~1
$ git checkout new_branch_name
```

Undoing recent changes

One of the main points of version control is to allow you to go back to any working state of your project and start over from there.

Get rid of all uncommited changes

```
$ git checkout .
```

Get rid of all changes since a specific commit

```
$ git reset --hard b9aedbb
```

Create new branch starting at a previous commit

```
$ git checkout -b branch_name b9aedbb
```

Stashing changes

If you want to save some changes without making a commit, you can stash your changes. This is useful when you want to revisit the most recent commit without making a new commit. You can stash as many sets of changes as you need.

Stash changes since last commit

```
$ git stash
Saved working directory and index state
    WIP on main: f6f39a6...
```

See stashed changes

```
$ git stash list
stash@{0}: WIP on main: f6f39a6...
stash@{1}: WIP on main: f6f39a6...
...
```

Reapply changes from most recent stash

```
$ git stash pop
```

Reapply changes from a specific stash

```
$ git stash pop --index 1
```

Clear all stashed changes

```
$ git stash clear
```

Comparing commits

It's often helpful to compare changes across different states of a project.

See all changes since last commit

```
$ git diff
```

See changes in one file since last commit

```
$ git diff hello.py
```

See changes since a specific commit

```
$ git diff HEAD~2
$ git diff HEAD^^
$ git diff fab2cdd
```

See changes between two commits

```
$ git diff fab2cdd 7c0a5d8
```

See changes in one file between two commits

```
$ git diff fab2cdd 7c0a5d8 hello.py
```

Good commit habits

Try to make a commit whenever your project is in a new working state. Make sure you're writing concise commit messages that focus on what changes have been implemented. If you're starting work on a new feature or bugfix, consider making a new branch.

Git & GitHub

GitHub is a platform for sharing code, and working collaboratively on code. You can clone any public project on GitHub. When you have an account, you can upload your own projects, and make them public or private.

Clone an existing repository to your local system

```
$ git clone
    https://github.com/ehmatthes/pcc_3e.git/
Cloning into 'pcc_3e'...
...
Resolving deltas: 100% (1503/1503), done.
```

Push a local project to a GitHub repository

You'll need to make an empty repository on GitHub first.

```
$ git remote add origin
  https://github.com/username/hello_repo.git
$ git push -u origin main
Enumerating objects: 10, done.
...
To https://github.com/username/hello_repo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch
    'main' from 'origin'.
```

Push recent changes to your GitHub repository

```
$ git push origin branch_name
```

Using pull requests

When you want to pull a set of changes from one branch into the main branch of a project on GitHub, you can make a pull request. To practice making pull requests on your own repositories, make a new branch for your work. When you're finished the work, push the branch to your repository. Then go to the "Pull requests" tab on GitHub, and click "Compare & pull request" on the branch you wish to merge. When you're ready, click "Merge pull request".

You can then pull these changes back into your local main branch with `git pull origin main`. This is an alternative to merging changes to your main branch locally, and then pushing the main branch to GitHub.

Practicing with Git

Git can be used in simple ways as a solo developer, and complex ways as part of a large collaborative team. You can gain valuable experience by making a simple throwaway project and trying all of these steps with that project. Make sure your project has multiple files and nested folders to get a clear sense of how Git works.