w4_git

Git

Configuring your identity

```
git config --global user.name "YOURNAME"
git config --global user.email "YOUREMAIL"
```

A sample project and repository

```
$ mkdir gitproject
$ cd gitproject
$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/kj24716/gitproject/.git/
$ git log
fatal: your current branch 'master' does not have any commits yet
$ echo -e "#include<stdio.h>\nint main(){\nprintf(\"Hi\");\nreturn 0;\n}\n" > test.c
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.c

nothing added to commit but untracked files present (use "git add" to track)
$ git add test.c
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   test.c

$ git commit -m "first file"
[master (root-commit) 4e87884] first file
 1 file changed, 6 insertions(+)
 create mode 100644 test.c
$ git status
On branch master
nothing to commit, working tree clean
$ git log
commit 4e878843247de9b4ced79406b1ea97c5f19b7564 (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:02:33 2024 +0000

    first file
```

Ignoring files

- Create a file `.gitignore` and add the single line `program` to it.
- Do another `git status` and notice that while the program is now ignored, the ignore file is marked as new. This file does belong in the repository, so add it and commit it.

```
$ gcc test.c -o test
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test

nothing added to commit but untracked files present (use "git add" to track)
$ echo "test" > .gitignore
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
$ git add .
$ git commit -m "ignore file"
[master bc20bbf] ignore file
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
$ git log
commit bc20bbf7ddd24faf04aeb44e34b22bdd2f9af259 (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:07:02 2024 +0000

    ignore file

commit 4e878843247de9b4ced79406b1ea97c5f19b7564
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:02:33 2024 +0000

    first file
```

Commit and checkout

Change *Hi* to *Hello* in the program, rebuild and run the program.

```
$ cat test.c | sed 's/Hi/Hello/' > test.c
$ git add .
$ git commit -m "Hi -> Hello"
[master 5e67efd] Hi -> Hello
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git log
commit 5e67efd414a9dbf1c80fa09b6f339cd520768b9e (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:16:36 2024 +0000

    Hi -> Hello

commit bc20bbf7ddd24faf04aeb44e34b22bdd2f9af259
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:07:02 2024 +0000

    ignore file
```

Sometimes you want to go back and look at another commit, or undo a commit that broke something—this is when you want a checkout.

- Note the first 6 or so characters of the commit hash of the commit where you added the ignore file, but before changing *Hi* to *Hello*. You need at least 6 characters, but only as many so that it's not ambiguous to git which commit you mean.
- Run `git checkout HASH` where HASH is the 6 or however many you need characters of the commit in question. Git will print a warning about the HEAD pointer.
- Check the source file, and notice that it is now back on *Hi*.
- Use `git checkout main` to return to the latest version of your files, and git will set up the HEAD pointer again ready to accept new commits.

```
$ git checkout bc20bb
Note: switching to 'bc20bb'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at bc20bbf ignore file
$ git checkout master
Previous HEAD position was bc20bbf ignore file
Switched to branch 'master'
```

HEAD is a pointer to the last commit you checked out: Either the last commit or the place you explicitly checked out with the git checkout command. Work you haven't committed can't be pointed to.

If you actually want to undo a commit, then you have two options:

- `git revert HASH` adds a new commit that returns the files to the state they were before the commit with the given hash. This is safe to use during team development, as it's just adding a new commit. If you have commits A, B and do `git revert B` then you get a new commit C so anyone else using the repository sees a sequence of commits A, B, C; but the state of the files in C is the same as in A.

```
$ git revert 5e67ef
[master 828eb92] Revert "Hi -> Hello"
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git log
commit 828eb929d0450811c00d1f9233bc7880acc77b86 (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:23:19 2024 +0000

    Revert "Hi -> Hello"

    This reverts commit 5e67efd414a9dbf1c80fa09b6f339cd520768b9e.

commit 5e67efd414a9dbf1c80fa09b6f339cd520768b9e
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:16:36 2024 +0000

    Hi -> Hello

$ cat test.c
#include<stdio.h>
int main(){
printf("Hi");
```

```
return 0;
}
```

- `git reset HASH` undoes commits by moving the HEAD pointer back to the commit with the given hash, but leaves the working copy alone (you can use the `-hard` option to change the files as well). This will break things if you have shared your newer commits with other developers, but it's safe to use to undo changes that you haven't pushed yet (we'll learn about this next time). The effect is as if the commits which you've reset had never happened.

```
$ git reset 5e67ef
Unstaged changes after reset:
M       test.c
$ 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:   test.c

no changes added to commit (use "git add" and/or "git commit -a")
$ git add .
$ git commit -m "reset to 5e67ef"
[master 30ba9c5] reset to 5e67ef
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git log
commit 30ba9c552322b0ce9e93d60a72a8dcf04d3bfc97 (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:26:11 2024 +0000

    reset to 5e67ef

commit 5e67efd414a9dbf1c80fa09b6f339cd520768b9e
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:16:36 2024 +0000

    Hi -> Hello

commit bc20bbf7ddd24faf04aeb44e34b22bdd2f9af259
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:07:02 2024 +0000

    ignore file
```

Git forges

Click the SSH tab and copy the URL there—it should be something like `git@github.com:USERNAME/REPONAME.git`.

```
$ ssh -T git@github.com
Hi saquantum! You've successfully authenticated, but GitHub does not provide shell access.
$ git clone git@github.com:saquantum/testgit.git
```

Go to that folder, and try `git remote show origin`.

```
$ git remote
origin
$ git remote -v
origin  git@github.com:saquantum/testgit.git (fetch)
origin  git@github.com:saquantum/testgit.git (push)
$ git remote show origin
* remote origin
  Fetch URL: git@github.com:saquantum/testgit.git
  Push  URL: git@github.com:saquantum/testgit.git
  HEAD branch: main
  Remote branches:
```

```
    develop   tracked
    main      tracked
    yzhbranch tracked
    zrbranch  tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)
```