The Git Challenge

Resources

Become familiar with the following online resources:

- ☐ The `git help` command (e.g. `git help clone`)
- ☐ book.git-scm.com (a nice intro text with inline videos)
- ☐ git-scm.com/documentation (links to web sites and videos)
- ☐ www.newartisans.com/2008/04/git-from-the-bottom-up.html (a PDF with an intermediate-level intro to Git's data model)

Basic Tasks

- ☐ Clone a repository by running
  `git clone git://github.com/git/hello-world.git`
- ☐ Edit python.py, replacing `World` with your name, and run `python python.py`
- ☐ Commit your change with `git add python.py` and `git commit` (pick a one-line "commit message" to describe your change)
- ☐ View the history in `gitk` to find the first 3 languages added:

- ☐ Merge my changes with
  `git pull git://aml.cs.byu.edu/amcnabb/hello-world.git master`
- ☐ Use `gitk` to identify which file I changed and when:

- ☐ Try to merge my next change with
  `git pull git://aml.cs.byu.edu/amcnabb/hello-conflict.git master` (this will fail because of a conflict with your change)
- ☐ Resolve this conflict by editing `python.py` and making a single line incorporating both your text and my text; don't forget to delete the 3 divider lines like `<<<<<<<` and any other leftover text
- ☐ Commit the merged file by running `git add python.py` and `git commit`
- ☐ Overwrite a file with `cp python.py java.java`
- ☐ Find what has changed with `git status` and `git diff --stat` and `git diff`
- ☐ Restore `java.java` by running `git checkout java.java` and look at the status and diff again
- ☐ Clone `git://github.com/schacon/gitbook` and browse its history in `gitk`
- ☐ Ask a question—give a friendly Git user a chance to help

Intermediate Tasks

Note: Some simple details are intentionally omitted for brevity. Also, tasks marked with the † symbol modify the commit history, and although they can be incredibly helpful for private editing, they must not be used to modify commits that other people may have already pulled.

- ☐ Clone Git's source code from `git://git.kernel.org/pub/scm/git/git.git`
- ☐ Stage two changes and reset one of them: change two files and run `git add` on both of them, run `git status` to see what has been staged, run `git reset` on one of the files, and then run `git status` again to see that one file is staged while the other file is modified but unstaged
- ☐ † Amend a commit by making and commiting a change with a typo in the commit message, then run `git commit --amend` to rewrite your commit message, and run `git log` to see that this replaced your previous commit (if you do a `git add` before the amend, then your new change will be included as part of the amended commit)
- ☐ Create a branch with `git checkout -b mybranch`, make a commit, switch back to the `master` branch with `git checkout master`, make another commit, switch back to `mybranch` with `git checkout mybranch`, make yet another commit, compare the branches with `git diff master`, switch back to the master branch, and then merge in `mybranch` with `git merge mybranch`, see what you did with `git log` or `gitk`, and finally delete the branch with `git branch -d mybranch`
- ☐ Make a partial commit: edit `README` and make two changes (one to the title and another at the end of the file), run `git diff` to see your changes, run `git add -p README` telling it `y` for the first "hunk" and `n` for the second hunk, then run `git diff --staged` to see that only part of the file has been staged, and then commit your partial change

☐ Create a centralized bare repository: create a directory called `sharedrepo.git` in a group-readable location, run `git --bare init --shared=group` and in that directory, and if you have a group configured run `chgrp -R mygroup .`, and from your personal repo push your data with `git push /path/to/sharedrepo.git master`

☐ Merge a branch with a conflict: create a `mybranch` branch, change a line in `mybranch` and commit, then change the same line in `master` and commit, perform a merge, and when the merge fails explore the difference between `git checkout --ours filename`, `git checkout --theirs filename`, and `git checkout -m filename`, and then resolve the conflict, and finally commit the merged file

☐ † Undo the most recent commit by running `git reset HEAD^`, use `git status` to see that the changes from this undone commit are preserved, run `git reflog` to see which commit you were on before and after your reset, and go back to the original commit with `git reset HEAD@{1}`

☐ † Undo the most recent commit with a hard reset by running `git reset --hard HEAD^`, use `git status` to see that the changes from this undone commit are destroyed, and go back to the original commit with `git reset HEAD@{1}`

☐ Revert a commit (creating a new commit that reverses the change): find out about a particular commit with `git log 7ec344^..7ec344` and `git diff 7ec344^..7ec344`, then perform the revert with `git revert 7ec344` and look at the results with `git log`

☐ Help another Git user solve a problem or find an answer

A few great Git commands have been neglected due to the awkwardness of creating a good simple example (contributions are welcomed). We recommend that you learn about `git cherry-pick`, `git stash`, and `git rebase`.

Advanced Tasks

☐ Pull out `contrib/examples` into its own repository. Hint: checkout a new `subdir` branch, run `git filter-branch --subdirectory-filter contrib/examples`, clone with `git clone -b subdir git git-examples`, and run `git branch -m subdir master` in the new repo

☐ Transplant a directory from one repo to another (from `git/Documentation` to `git-examples/docs`). Hint: in the `git` repo, create a new `transplant` branch, isolate the `Documentation` directory with `subdirectory-filter` as above, rename to the desired path with `git filter-branch --tree-filter "mkdir -p docs; git mv -k * docs"`, and pull the `transplant` branch into `git-examples`

☐ † Permanently remove the `t` (tests) directory from the history. Hint: `git filter-branch --index-filter "git rm -r --cached --ignore-unmatch t" --prune-empty` (note that this will require removing and recloning any copy of your repo)

☐ Rewrite the previous command using `tree-filter` instead of `index-filter` (and consider which is better and why):

☐ Add a commit to a bare repository (useful for scripting). Hint: create a blob and get its id with `git hash-object -w /path/to/file.txt`, find the current head with `git show-ref --heads --hash master`, read the tree into an index file with `GIT_INDEX_FILE=/tmp/git_index git read-tree "$head"`, add the ref of the blob into the index file with `GIT_INDEX_FILE=/tmp/git_index git update-index --add --cacheinfo 100644 "$blob" path/within/repo/file.txt`, write the index to the repo and get the tree id with `GIT_INDEX_FILE=/tmp/git_index git write-tree`, write a commit and get the commit id with `echo "commit message here" |git commit-tree "$tree" -p "$head"`, and finally update the head of the repository to be the new commit with `git update-ref refs/heads/master "$commit" "$head"`

☐ Find out what `git/contrib/examples` is for and browse one of its scripts

☐ Create a new Git task to teach a concept or solve a problem