CSG - Tips and Techniques

Duncan C. White

CSG Systems Manager,
Dept of Computing,
Imperial College London

Oct 2012

Today, I want to tell you a little about CSG - the **Computing Support Group** - and take a quick run through a number of useful tips and techniques to let you **work smarter**.
The topics I'll look at are the following:

- Introduction to CSG.
- Reporting problems (the smart way).

Today, I want to tell you a little about CSG - the **Computing Support Group** - and take a quick run through a number of useful tips and techniques to let you **work smarter**.
The topics I'll look at are the following:

- Introduction to CSG.
- Reporting problems (the smart way).
- Passwords and security.
- Working with directories.
- Recovering lost files from /vol/recover.

Today, I want to tell you a little about CSG - the **Computing Support Group** - and take a quick run through a number of useful tips and techniques to let you **work smarter**.
The topics I'll look at are the following:

- Introduction to CSG.
- Reporting problems (the smart way).
- Passwords and security.
- Working with directories.
- Recovering lost files from /vol/recover.
- Backups and version control.
- Advanced features of the shell.

Today, I want to tell you a little about CSG - the **Computing Support Group** - and take a quick run through a number of useful tips and techniques to let you **work smarter**.
The topics I'll look at are the following:

- Introduction to CSG.
- Reporting problems (the smart way).
- Passwords and security.
- Working with directories.
- Recovering lost files from /vol/recover.
- Backups and version control.
- Advanced features of the shell.
- Unix pipelines and filters.
- A little information about the Perl scripting language and my upcoming Perl course.
- How to find out more information.

- The **Computing Support Group (CSG)** provides Computing support solely to DoC.

- The **Computing Support Group (CSG)** provides Computing support solely to DoC.
- A separate organisation – **Information and Communication Technologies (ICT)** – covers the rest of College.
- We now have an ICT support engineer (**Niels**) based with us, so you can come in and ask him about any ICT-related problems.

- The **Computing Support Group (CSG)** provides Computing support solely to DoC.
- A separate organisation – **Information and Communication Technologies (ICT)** – covers the rest of College.
- We now have an ICT support engineer (**Niels**) based with us, so you can come in and ask him about any ICT-related problems.
- Not sure who to ask? Lab exercise problems - Lab organisers. Teaching database/CATE problems - `cate@doc.ic.ac.uk`. Mac problems, wifi problems, printing problems, PCs not in DoC - ICT (visit Niels in 225, **phone 0207 594 9000, email** `service.desk@imperial.ac.uk`), other computing support questions - us.

- The **Computing Support Group (CSG)** provides Computing support solely to DoC.
- A separate organisation – **Information and Communication Technologies (ICT)** – covers the rest of College.
- We now have an ICT support engineer (**Niels**) based with us, so you can come in and ask him about any ICT-related problems.
- Not sure who to ask? Lab exercise problems - Lab organisers. Teaching database/CATE problems - `cate@doc.ic.ac.uk`. Mac problems, wifi problems, printing problems, PCs not in DoC - ICT (visit Niels in 225, **phone 0207 594 9000, email** `service.desk@imperial.ac.uk`), other computing support questions - us.
- Please report new problems (or progress existing ones) using the CSG problem tracking web page:

  `https://otrs.doc.ic.ac.uk/`

- The **Computing Support Group (CSG)** provides Computing support solely to DoC.
- A separate organisation – **Information and Communication Technologies (ICT)** – covers the rest of College.
- We now have an ICT support engineer (**Niels**) based with us, so you can come in and ask him about any ICT-related problems.
- Not sure who to ask? Lab exercise problems - Lab organisers. Teaching database/CATE problems - `cate@doc.ic.ac.uk`. Mac problems, wifi problems, printing problems, PCs not in DoC - ICT (visit Niels in 225, **phone 0207 594 9000, email** `service.desk@imperial.ac.uk`), other computing support questions - us.
- Please report new problems (or progress existing ones) using the CSG problem tracking web page:

  `https://otrs.doc.ic.ac.uk/`
- Or email `help@doc.ic.ac.uk` (aka `doc-help@imperial.ac.uk`).

- The **Computing Support Group (CSG)** provides Computing support solely to DoC.
- A separate organisation – **Information and Communication Technologies (ICT)** – covers the rest of College.
- We now have an ICT support engineer (**Niels**) based with us, so you can come in and ask him about any ICT-related problems.
- Not sure who to ask? Lab exercise problems - Lab organisers. Teaching database/CATE problems - `cate@doc.ic.ac.uk`. Mac problems, wifi problems, printing problems, PCs not in DoC - ICT (visit Niels in 225, **phone 0207 594 9000, email** `service.desk@imperial.ac.uk`), other computing support questions - us.
- Please report new problems (or progress existing ones) using the CSG problem tracking web page:

  `https://otrs.doc.ic.ac.uk/`
- Or email `help@doc.ic.ac.uk` (aka `doc-help@imperial.ac.uk`).
- Or come in person directly to room 225 Huxley.

- Please report problems to us - many problems are never reported!
- When reporting a problem, put yourself in our shoes - what information will we obviously need? Have this with you!

- Please report problems to us - many problems are never reported!
- When reporting a problem, put yourself in our shoes - what information will we obviously need? Have this with you!
- Example: *A PC in the lab is broken.* Obvious questions: *Which specific PC*? (eg. matrix17). *Broken how*? (eg. no power, power but no video, freeze at bios, etc).
- Example: *firefox just crashed*. Obvious data we need: your username, name of machine, operating system in use (eg. Linux), what you did, any error message that appeared.

- Please report problems to us - many problems are never reported!
- When reporting a problem, put yourself in our shoes - what information will we obviously need? Have this with you!
- Example: *A PC in the lab is broken.* Obvious questions: *Which specific PC*? (eg. matrix17). *Broken how*? (eg. no power, power but no video, freeze at bios, etc).
- Example: *firefox just crashed*. Obvious data we need: your username, name of machine, operating system in use (eg. Linux), what you did, any error message that appeared.
- In essence: describe the problem clearly.

- Please report problems to us - many problems are never reported!
- When reporting a problem, put yourself in our shoes - what information will we obviously need? Have this with you!
- Example: *A PC in the lab is broken.* Obvious questions: *Which specific PC*? (eg. matrix17). *Broken how*? (eg. no power, power but no video, freeze at bios, etc).
- Example: *firefox just crashed*. Obvious data we need: your username, name of machine, operating system in use (eg. Linux), what you did, any error message that appeared.
- In essence: describe the problem clearly.
- Try logging into a second PC, see what happens there. Does the problem follow you or stay with a particular lab PC?
- Read Eric S. Raymond's superb *How To Ask Questions The Smart Way* article at: `http://catb.org/~esr/faqs/smart-questions.html`

- Goal: each user should have a single College password. DoC Linux machines use your College password directly. DoC Windows machines store passwords separately - but changing your College password via ICT's password change webpage **also tries to set** your DoC Windows password.

- Goal: each user should have a single College password. DoC Linux machines use your College password directly. DoC Windows machines store passwords separately - but changing your College password via ICT's password change webpage **also tries to set** your DoC Windows password.
- So: your College and Windows passwords are usually the same. If the synchronization fails (sigh), manually change your DoC Windows password or ask us.

- Goal: each user should have a single College password. DoC Linux machines use your College password directly. DoC Windows machines store passwords separately - but changing your College password via ICT's password change webpage **also tries to set** your DoC Windows password.
- So: your College and Windows passwords are usually the same. If the synchronization fails (sigh), manually change your DoC Windows password or ask us.
- Default assumption: **You are personally responsible for everything done using your account.**

- Goal: each user should have a single College password. DoC Linux machines use your College password directly. DoC Windows machines store passwords separately - but changing your College password via ICT's password change webpage **also tries to set** your DoC Windows password.
- So: your College and Windows passwords are usually the same. If the synchronization fails (sigh), manually change your DoC Windows password or ask us.
- Default assumption: **You are personally responsible for everything done using your account.**
- **Never** tell anyone else, even your best friend, your password. Do not send your password across any network in plain text - eg. via email or http://. Don't even tell a member of CSG or ICT your password - we won't **ever** ask.

- Goal: each user should have a single College password. DoC Linux machines use your College password directly. DoC Windows machines store passwords separately - but changing your College password via ICT's password change webpage **also tries to set** your DoC Windows password.
- So: your College and Windows passwords are usually the same. If the synchronization fails (sigh), manually change your DoC Windows password or ask us.
- Default assumption: **You are personally responsible for everything done using your account.**
- **Never** tell anyone else, even your best friend, your password. Do not send your password across any network in plain text - eg. via email or http://. Don't even tell a member of CSG or ICT your password - we won't **ever** ask.
- Passwords are still the cornerstone of computer and network security, so: choose **strong passwords** and change them often.

- Strong passwords should be at least 10-20 characters and use digits, punctuation symbols, lowercase and uppercase letters.

- Strong passwords should be at least 10-20 characters and use digits, punctuation symbols, lowercase and uppercase letters.
- Ideally, strong passwords look like **random line noise**!

- Strong passwords should be at least 10-20 characters and use digits, punctuation symbols, lowercase and uppercase letters.
- Ideally, strong passwords look like **random line noise**!
- **Don't** use a letter-sequence followed by a digit-sequence followed by a punctuation sequence - **mix it up**.
- **Confound** expectations - put uppercase letters in the middle or end of the password!

- Strong passwords should be at least 10-20 characters and use digits, punctuation symbols, lowercase and uppercase letters.
- Ideally, strong passwords look like **random line noise**!
- **Don't** use a letter-sequence followed by a digit-sequence followed by a punctuation sequence - **mix it up**.
- **Confound** expectations - put uppercase letters in the middle or end of the password!
- Generate **many passwords** from a single scheme, for instance:
  - Choose a **phrase**, such as `Plan 9 from Outer Space`.

- Strong passwords should be at least 10-20 characters and use digits, punctuation symbols, lowercase and uppercase letters.
- Ideally, strong passwords look like **random line noise**!
- **Don't** use a letter-sequence followed by a digit-sequence followed by a punctuation sequence - **mix it up**.
- **Confound** expectations - put uppercase letters in the middle or end of the password!
- Generate **many passwords** from a single scheme, for instance:
  - Choose a **phrase**, such as `Plan 9 from Outer Space`.
  - Take the **initial letters** of the phrase – `p9fos`.

- Strong passwords should be at least 10-20 characters and use digits, punctuation symbols, lowercase and uppercase letters.
- Ideally, strong passwords look like **random line noise**!
- **Don't** use a letter-sequence followed by a digit-sequence followed by a punctuation sequence - **mix it up**.
- **Confound** expectations - put uppercase letters in the middle or end of the password!
- Generate **many passwords** from a single scheme, for instance:
    - Choose a **phrase**, such as `Plan 9 from Outer Space`.
    - Take the **initial letters** of the phrase – `p9fos`.
    - Randomly **uppercase a few letters** in the phrase – `p9FoS`.

- Strong passwords should be at least 10-20 characters and use digits, punctuation symbols, lowercase and uppercase letters.
- Ideally, strong passwords look like **random line noise**!
- **Don't** use a letter-sequence followed by a digit-sequence followed by a punctuation sequence - **mix it up**.
- **Confound** expectations - put uppercase letters in the middle or end of the password!
- Generate **many passwords** from a single scheme, for instance:
    - Choose a **phrase**, such as `Plan 9 from Outer Space`.
    - Take the **initial letters** of the phrase – `p9fos`.
    - Randomly **uppercase a few letters** in the phrase – `p9FoS`.
    - Add **digits and punctuation at random** until long enough: `p!9F,4oS1`.

- Strong passwords should be at least 10-20 characters and use digits, punctuation symbols, lowercase and uppercase letters.
- Ideally, strong passwords look like **random line noise**!
- **Don't** use a letter-sequence followed by a digit-sequence followed by a punctuation sequence - **mix it up**.
- **Confound** expectations - put uppercase letters in the middle or end of the password!
- Generate **many passwords** from a single scheme, for instance:
  - Choose a **phrase**, such as `Plan 9 from Outer Space`.
  - Take the **initial letters** of the phrase – `p9fos`.
  - Randomly **uppercase a few letters** in the phrase – `p9FoS`.
  - Add **digits and punctuation at random** until long enough: `p!9F,4oS1`.
- Check potential **password schemes** via password strength checkers: `http://www.passwordmeter.com/`

- Strong passwords should be at least 10-20 characters and use digits, punctuation symbols, lowercase and uppercase letters.
- Ideally, strong passwords look like **random line noise**!
- **Don't** use a letter-sequence followed by a digit-sequence followed by a punctuation sequence - **mix it up**.
- **Confound** expectations - put uppercase letters in the middle or end of the password!
- Generate **many passwords** from a single scheme, for instance:
    - Choose a **phrase**, such as `Plan 9 from Outer Space`.
    - Take the **initial letters** of the phrase – `p9fos`.
    - Randomly **uppercase a few letters** in the phrase – `p9FoS`.
    - Add **digits and punctuation at random** until long enough: `p!9F,4oS1`.
- Check potential **password schemes** via password strength checkers: `http://www.passwordmeter.com/`
- **don't check your actual password!!!**

- Hopefully you already know how to create directories with `mkdir` and navigate the directory structure with `cd` and `ls`. If not, find a tutorial by Googling for "introduction to Linux shell".

- Hopefully you already know how to create directories with `mkdir` and navigate the directory structure with `cd` and `ls`. If not, find a tutorial by Googling for "introduction to Linux shell".
- Think seriously about a sensible directory structure.
- For example, tidy up each lab exercise into a separate directory, perhaps under a "cs1-labs" directory.

- Hopefully you already know how to create directories with `mkdir` and navigate the directory structure with `cd` and `ls`. If not, find a tutorial by Googling for "introduction to Linux shell".
- Think seriously about a sensible directory structure.
- For example, tidy up each lab exercise into a separate directory, perhaps under a "cs1-labs" directory.
- When you finish an exercise, think about what files you need to keep. Delete as much as possible - but not too much!
- For instance, if you happened to be programming in C, keep source files (.c and .h files) and compilation instructions (eg. Makefiles) but delete all generated files - eg. `rm *.o` etc. If you're using `make` then setup `make clean` to delete all the rubbish.

- Hopefully you already know how to create directories with `mkdir` and navigate the directory structure with `cd` and `ls`. If not, find a tutorial by Googling for "introduction to Linux shell".
- Think seriously about a sensible directory structure.
- For example, tidy up each lab exercise into a separate directory, perhaps under a "cs1-labs" directory.
- When you finish an exercise, think about what files you need to keep. Delete as much as possible - but not too much!
- For instance, if you happened to be programming in C, keep source files (.c and .h files) and compilation instructions (eg. Makefiles) but delete all generated files - eg. `rm *.o` etc. If you're using `make` then setup `make clean` to delete all the rubbish.
- Later on, tar and gzip each exercise directory into a tarball.

- Hopefully you already know how to create directories with `mkdir` and navigate the directory structure with `cd` and `ls`. If not, find a tutorial by Googling for "introduction to Linux shell".
- Think seriously about a sensible directory structure.
- For example, tidy up each lab exercise into a separate directory, perhaps under a "cs1-labs" directory.
- When you finish an exercise, think about what files you need to keep. Delete as much as possible - but not too much!
- For instance, if you happened to be programming in C, keep source files (.c and .h files) and compilation instructions (eg. Makefiles) but delete all generated files - eg. `rm *.o` etc. If you're using `make` then setup `make clean` to delete all the rubbish.
- Later on, tar and gzip each exercise directory into a tarball.
- This technique can make a real difference in reducing your use of home directory disk space....

- You can store up to 4GB (and 20000 files) in your DoC home directory (drive **H:** in Windows). The limits are enforced by the Unix **quota** system.

- You can store up to 4GB (and 20000 files) in your DoC home directory (drive **H:** in Windows). The limits are enforced by the Unix **quota** system.
- To check how much disk space you are using - use the command: `quota -Q`.
- Please check your quota at least once a week. When over quota, start freeing up space immediately!

- You can store up to 4GB (and 20000 files) in your DoC home directory (drive **H:** in Windows). The limits are enforced by the Unix **quota** system.
- To check how much disk space you are using - use the command: `quota -Q`.
- Please check your quota at least once a week. When over quota, start freeing up space immediately!
- Read `www.doc.ic.ac.uk/csg/guides/file-storage/quota` for lots of quota-related tips and tricks (such as how to find where you're using the space or files).

- You can store up to 4GB (and 20000 files) in your DoC home directory (drive **H:** in Windows). The limits are enforced by the Unix **quota** system.
- To check how much disk space you are using - use the command: `quota -Q`.
- Please check your quota at least once a week. When over quota, start freeing up space immediately!
- Read `www.doc.ic.ac.uk/csg/guides/file-storage/quota` for lots of quota-related tips and tricks (such as how to find where you're using the space or files).
- If you really need extra quota, please email `help@doc.ic.ac.uk`...

- You can store up to 4GB (and 20000 files) in your DoC home directory (drive **H:** in Windows). The limits are enforced by the Unix **quota** system.
- To check how much disk space you are using - use the command: `quota -Q`.
- Please check your quota at least once a week. When over quota, start freeing up space immediately!
- Read `www.doc.ic.ac.uk/csg/guides/file-storage/quota` for lots of quota-related tips and tricks (such as how to find where you're using the space or files).
- If you really need extra quota, please email `help@doc.ic.ac.uk`...
- ... in the smart way: providing an estimate of how much extra space you need, how long you'll need it for, and why!

- You can store up to 4GB (and 20000 files) in your DoC home directory (drive **H:** in Windows). The limits are enforced by the Unix **quota** system.
- To check how much disk space you are using - use the command: `quota -Q`.
- Please check your quota at least once a week. When over quota, start freeing up space immediately!
- Read `www.doc.ic.ac.uk/csg/guides/file-storage/quota` for lots of quota-related tips and tricks (such as how to find where you're using the space or files).
- If you really need extra quota, please email `help@doc.ic.ac.uk`...
- ... in the smart way: providing an estimate of how much extra space you need, how long you'll need it for, and why!
- You can store data in a shared area `/vol/bitbucket`, newly expanded to 10TB, and a private area on each PC called `/data` (ideal for storing a .iso file before burning a DVD). **BUT: bitbucket and /data are NOT backed up!**

- We provide an online backup system with nightly snapshots of your home directory - letting you recover files yourself.

- We provide an online backup system with nightly snapshots of your home directory - letting you recover files yourself.
- Suppose I delete my C-shell `.cshrc` file. Ooops!

- We provide an online backup system with nightly snapshots of your home directory - letting you recover files yourself.
- Suppose I delete my C-shell `.cshrc` file. Ooops!
- Don't panic! `cd /vol/recover/homes/USERNAME` and `ls`: directories called `full` and `inc-00..12`.
- `full` is a copy of your home dir from **last night**, while `inc-NN` dirs contain *backwards incremental* changes.

  To see what versions of my lost file are available:

  ```
  ls -l */.cshrc
  ```

- We provide an online backup system with nightly snapshots of your home directory - letting you recover files yourself.
- Suppose I delete my C-shell `.cshrc` file. Ooops!
- Don't panic! `cd /vol/recover/homes/USERNAME` and `ls`: directories called `full` and `inc-00..12`.
- `full` is a copy of your home dir from **last night**, while `inc-NN` dirs contain *backwards incremental* changes.
  To see what versions of my lost file are available:

  ```
  ls -l */.cshrc
  ```

- Now I can look at each version, and choose a particular one. Because I destroyed my `.cshrc` file just now, the `full/.cshrc` is the one I want. I can then copy it back into my home directory:

  ```
  cp -p full/.cshrc ~/.cshrc
  ```

- We provide an online backup system with nightly snapshots of your home directory - letting you recover files yourself.
- Suppose I delete my C-shell `.cshrc` file. Ooops!
- Don't panic! `cd /vol/recover/homes/USERNAME` and `ls`: directories called `full` and `inc-00..12`.
- `full` is a copy of your home dir from **last night**, while `inc-NN` dirs contain *backwards incremental* changes.
  To see what versions of my lost file are available:

  ```
  ls -l */.cshrc
  ```

- Now I can look at each version, and choose a particular one. Because I destroyed my `.cshrc` file just now, the `full/.cshrc` is the one I want. I can then copy it back into my home directory:

  ```
  cp -p full/.cshrc ~/.cshrc
  ```

- See `www.doc.ic.ac.uk/csg/guides/file-storage/backups`.
- Note: Your Windows profile (eg. the desktop, my documents and the application data folder) is not backed up. In Windows, only store important stuff on drive `H:`.

- Try not to lose important files in the first place!

- Try not to lose important files in the first place!
- Simple ways - you're working in a single project directory, tar and gzip the entire directory every so often - eg.

  ```
  tar czf ~/backups/proj-date-time.tgz proj-dir
  ```

- Try not to lose important files in the first place!
- Simple ways - you're working in a single project directory, tar and gzip the entire directory every so often - eg.

  `tar czf ~/backups/proj-date-time.tgz proj-dir`
- However: this is not a new problem. Do backups the smart way: learn to use a version control system, and use it routinely!

- Try not to lose important files in the first place!
- Simple ways - you're working in a single project directory, tar and gzip the entire directory every so often - eg.

  ```
  tar czf ~/backups/proj-date-time.tgz proj-dir
  ```
- However: this is not a new problem. Do backups the smart way: learn to use a version control system, and use it routinely!
- On our Linux systems, there are at least 4 version control systems available: RCS, CVS, Subversion (SVN) and GIT. We recommend GIT - Tristan is providing you with notes on using git already. We strongly support Tristan's initialive, and encourage you to use git routinely for your own projects.

- Try not to lose important files in the first place!
- Simple ways - you're working in a single project directory, tar and gzip the entire directory every so often - eg.

  `tar czf ~/backups/proj-date-time.tgz proj-dir`
- However: this is not a new problem. Do backups the smart way: learn to use a version control system, and use it routinely!
- On our Linux systems, there are at least 4 version control systems available: RCS, CVS, Subversion (SVN) and GIT. We recommend GIT - Tristan is providing you with notes on using git already. We strongly support Tristan's initialive, and encourage you to use git routinely for your own projects.
- You can also read our version control guides:

  `www.doc.ic.ac.uk/csg/guides/version-control`

- When typing a filename into the shell, once you have typed enough to uniquely select a single file, press the `TAB` key and the shell will complete the filename for you.

- When typing a filename into the shell, once you have typed enough to uniquely select a single file, press the `TAB` key and the shell will complete the filename for you.
- If there are several alternatives, `CTRL-D` will list all the matching files, and then redisplay the partial command line.
- The shell (as setup for you) remembers the previous 500 commands you have previously typed, and allows you to reissue them. It can even save history between sessions if asked.

- When typing a filename into the shell, once you have typed enough to uniquely select a single file, press the `TAB` key and the shell will complete the filename for you.
- If there are several alternatives, `CTRL-D` will list all the matching files, and then redisplay the partial command line.

- The shell (as setup for you) remembers the previous 500 commands you have previously typed, and allows you to reissue them. It can even save history between sessions if asked.
- Type `history [N]` to see the whole list of (or last N) remembered commands. `h` has been aliased to `history 15`.
- To move through the previous few commands, press the up-arrow key to go back, and the down-arrow key to go forward.

- When typing a filename into the shell, once you have typed enough to uniquely select a single file, press the `TAB` key and the shell will complete the filename for you.
- If there are several alternatives, `CTRL-D` will list all the matching files, and then redisplay the partial command line.

- The shell (as setup for you) remembers the previous 500 commands you have previously typed, and allows you to reissue them. It can even save history between sessions if asked.
- Type `history [N]` to see the whole list of (or last N) remembered commands. `h` has been aliased to `history 15`.
- To move through the previous few commands, press the up-arrow key to go back, and the down-arrow key to go forward.
- Alternatively, to redo command 200 in the history list, type `!200`
- There are many `!` commands, read `man tcsh` for more details.

- If you want to redo *the last vim command*, type `!vim`, `!vi` or even `!v`.

- If you want to redo *the last vim command*, type `!vim`, `!vi` or even `!v`.
- You often perform several successive operations on a single file, `!$` means the *last argument of the last command*:

```
vim ~/cs1-labs/ex4/factorial.hs
grep fact !$
cp !$ ~/backups
```

- If you want to redo *the last vim command*, type `!vim`, `!vi` or even `!v`.
- You often perform several successive operations on a single file, `!$` means the *last argument of the last command*:

```
vim ~/cs1-labs/ex4/factorial.hs
grep fact !$
cp !$ ~/backups
```

- Having typed an `!` command, you may wish to see the command before running it - perhaps edit it! Having typed any of:

```
cp !$ ~/backups
!vim
!200
```

- If you want to redo *the last vim command*, type `!vim`, `!vi` or even `!v`.
- You often perform several successive operations on a single file, `!$` means the *last argument of the last command*:

```
vim ~/cs1-labs/ex4/factorial.hs
grep fact !$
cp !$ ~/backups
```

- Having typed an `!` command, you may wish to see the command before running it - perhaps edit it! Having typed any of:

```
cp !$ ~/backups
!vim
!200
```

  type `ESC-spacebar` (press and release the ESC key, then press and release the spacebar), and the `!` command will be expanded.

- If you want to redo *the last vim command*, type `!vim`, `!vi` or even `!v`.
- You often perform several successive operations on a single file, `!$` means the *last argument of the last command*:

```
vim ~/cs1-labs/ex4/factorial.hs
grep fact !$
cp !$ ~/backups
```

- Having typed an `!` command, you may wish to see the command before running it - perhaps edit it! Having typed any of:

```
cp !$ ~/backups
!vim
!200
```

  type `ESC-spacebar` (press and release the ESC key, then press and release the spacebar), and the `!` command will be expanded.
- Another option: type `vim` and then type `ESC-p`. Up pops the last `vim` command - type `ESC-p` again and up pops the previous `vim` command etc.

- Often, you work in a single directory for a while, then want to *nip over* to another directory, look around for a while, then *return* to the first directory.

- Often, you work in a single directory for a while, then want to *nip over* to another directory, look around for a while, then *return* to the first directory.
- The shell has the ability to remember not just the *current directory* but a *stack of directories you're still interested in*. The current directory is really just the top of the directory stack!
- To push a directory onto the stack, simply replace `cd` with `pushd`, as in `pushd ~/cs1-labs/ex4`.

- Often, you work in a single directory for a while, then want to *nip over* to another directory, look around for a while, then *return* to the first directory.
- The shell has the ability to remember not just the *current directory* but a *stack of directories you're still interested in*. The current directory is really just the top of the directory stack!
- To `push` a directory onto the stack, simply replace `cd` with `pushd`, as in `pushd ~/cs1-labs/ex4`.
- To ask what's on the directory stack, type `dirs`.

- Often, you work in a single directory for a while, then want to *nip over* to another directory, look around for a while, then *return* to the first directory.
- The shell has the ability to remember not just the *current directory* but a *stack of directories you're still interested in*. The current directory is really just the top of the directory stack!
- To `push` a directory onto the stack, simply replace `cd` with `pushd`, as in `pushd ~/cs1-labs/ex4`.
- To ask what's on the directory stack, type `dirs`.
- To pop the current directory off, returning to the previous directory, simply type `popd`. Alternatively, you might want to stay in the current directory, and pop off the previous directory, that's `popd +1`.

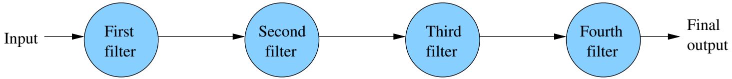
- Often, you work in a single directory for a while, then want to *nip over* to another directory, look around for a while, then *return* to the first directory.
- The shell has the ability to remember not just the *current directory* but a *stack of directories you're still interested in*. The current directory is really just the top of the directory stack!
- To `push` a directory onto the stack, simply replace `cd` with `pushd`, as in `pushd ~/cs1-labs/ex4`.
- To ask what's on the directory stack, type `dirs`.
- To pop the current directory off, returning to the previous directory, simply type `popd`. Alternatively, you might want to stay in the current directory, and pop off the previous directory, that's `popd +1`.
- To swap the top two directories - ie. to return to the previous directory, putting the current directory onto the directory stack as the previous, use `pushd` with no argument.
- Can also `pushd +N` - swap the current directory and the Nth directory down the stack.

- Much of the power of Unix - when you get below the glitzy Gnome graphical user interface - comes from the **toolkit** of small but powerful programs, called *filters*, that can be linked together.

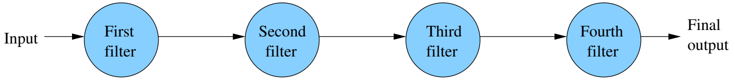
- Much of the power of Unix - when you get below the glitzy Gnome graphical user interface - comes from the **toolkit** of small but powerful programs, called *filters*, that can be linked together.
- Each filter reads input from the standard input *(stdin)* and/or named files, performs some simple sorting, searching or summarising operation on the data, and then writes the resulting output to the standard output *(stdout)*.

- Much of the power of Unix - when you get below the glitzy Gnome graphical user interface - comes from the **toolkit** of small but powerful programs, called *filters*, that can be linked together.
- Each filter reads input from the standard input *(stdin)* and/or named files, performs some simple sorting, searching or summarising operation on the data, and then writes the resulting output to the standard output *(stdout).*
- Filters can be linked together into a *pipeline*, in which the *output* from one filter can be provided as the *input* to another filter.

- Much of the power of Unix - when you get below the glitzy Gnome graphical user interface - comes from the **toolkit** of small but powerful programs, called *filters*, that can be linked together.
- Each filter reads input from the standard input *(stdin)* and/or named files, performs some simple sorting, searching or summarising operation on the data, and then writes the resulting output to the standard output *(stdout).*
- Filters can be linked together into a *pipeline*, in which the *output* from one filter can be provided as the *input* to another filter.
- Input flows into the pipeline, is processed by each stage of the pipeline, and output emerges from the end of the pipeline:



- Much of the power of Unix - when you get below the glitzy Gnome graphical user interface - comes from the **toolkit** of small but powerful programs, called *filters*, that can be linked together.
- Each filter reads input from the standard input *(stdin)* and/or named files, performs some simple sorting, searching or summarising operation on the data, and then writes the resulting output to the standard output *(stdout)*.
- Filters can be linked together into a *pipeline*, in which the *output* from one filter can be provided as the *input* to another filter.
- Input flows into the pipeline, is processed by each stage of the pipeline, and output emerges from the end of the pipeline:



- *Shell Programming* describes the way in which a series of Unix filter pipelines, plus loops and conditionals, may be placed into a file and made executable, forming a *Shell Script*.

Some filters are single-function, whereas others are considerably more open ended. Some of the most important are:

- **wc** - read each file of input, counting words, characters and lines.

Some filters are single-function, whereas others are considerably more open ended. Some of the most important are:

- **wc** - read each file of input, counting words, characters and lines.
- **tr** - translate every occurrence of any character in one set of characters into the corresponding character in another set. Given only one set, deletes all occurrences of all characters in the set.

Some filters are single-function, whereas others are considerably more open ended. Some of the most important are:

- **wc** - read each file of input, counting words, characters and lines.
- **tr** - translate every occurrence of any character in one set of characters into the corresponding character in another set. Given only one set, deletes all occurrences of all characters in the set.
- **sort** - sort the entire input, line by line, in a variety of sorting orders (ascending, descending, numeric, alphabetic etc).

Some filters are single-function, whereas others are considerably more open ended. Some of the most important are:

- **wc** - read each file of input, counting words, characters and lines.
- **tr** - translate every occurrence of any character in one set of characters into the corresponding character in another set. Given only one set, deletes all occurrences of all characters in the set.
- **sort** - sort the entire input, line by line, in a variety of sorting orders (ascending, descending, numeric, alphabetic etc).
- **grep** - extract all the lines from the input that match a particular pattern called a **regular expression**.

Some filters are single-function, whereas others are considerably more open ended. Some of the most important are:

- **wc** - read each file of input, counting words, characters and lines.
- **tr** - translate every occurrence of any character in one set of characters into the corresponding character in another set. Given only one set, deletes all occurrences of all characters in the set.
- **sort** - sort the entire input, line by line, in a variety of sorting orders (ascending, descending, numeric, alphabetic etc).
- **grep** - extract all the lines from the input that match a particular pattern called a **regular expression**.
- **sed** - a stream editor that lets you perform a variety of structural changes to the input stream. You can delete particular lines (either by range of line numbers, or all those lines matching a regular expression), search and replace lines, add new lines, extract particular lines etc.

Some filters are single-function, whereas others are considerably more open ended. Some of the most important are:

- **wc** - read each file of input, counting words, characters and lines.
- **tr** - translate every occurrence of any character in one set of characters into the corresponding character in another set. Given only one set, deletes all occurrences of all characters in the set.
- **sort** - sort the entire input, line by line, in a variety of sorting orders (ascending, descending, numeric, alphabetic etc).
- **grep** - extract all the lines from the input that match a particular pattern called a **regular expression**.
- **sed** - a stream editor that lets you perform a variety of structural changes to the input stream. You can delete particular lines (either by range of line numbers, or all those lines matching a regular expression), search and replace lines, add new lines, extract particular lines etc.
- **awk** - an interpreted pattern-matching language which treats each line as a record, allows you to process a set of records, selecting and summarising some fields of some of the records.

- **Perl** is another interpreted scripting language - far better than **awk** - good enough to be used to develop full large-scale computer systems, or just used via a "Perl one-liner" on the command line.

- **Perl** is another interpreted scripting language - far better than **awk** - good enough to be used to develop full large-scale computer systems, or just used via a "Perl one-liner" on the command line.
- I run a short **Perl Course** in week 11 (late December), mainly for second years - but everyone's welcome. The notes will soon be online: `http://www.doc.ic.ac.uk/~dcw/perl2012/`

- **Perl** is another interpreted scripting language - far better than **awk** - good enough to be used to develop full large-scale computer systems, or just used via a "Perl one-liner" on the command line.
- I run a short **Perl Course** in week 11 (late December), mainly for second years - but everyone's welcome. The notes will soon be online: `http://www.doc.ic.ac.uk/~dcw/perl2012/`
- Let's see a few examples: One commonly needed filter that doesn't exist is *sum up all these numbers*. The canonical way of doing this used to be via **awk**:

```
du -s * | awk '{t += $1} END {print t}'
```

- **Perl** is another interpreted scripting language - far better than **awk** - good enough to be used to develop full large-scale computer systems, or just used via a "Perl one-liner" on the command line.
- I run a short **Perl Course** in week 11 (late December), mainly for second years - but everyone's welcome. The notes will soon be online: `http://www.doc.ic.ac.uk/~dcw/perl2012/`
- Let's see a few examples: One commonly needed filter that doesn't exist is *sum up all these numbers*. The canonical way of doing this used to be via **awk**:

```
du -s * | awk '{t += $1} END {print t}'
```

- **Perl's** equivalent is a few characters longer and shows clearly how **awk** was a major element in the Perl language design:

```
du -s * | perl -nle '$t += $_; END {print $t}'
```

- **Perl** is another interpreted scripting language - far better than **awk** - good enough to be used to develop full large-scale computer systems, or just used via a "Perl one-liner" on the command line.
- I run a short **Perl Course** in week 11 (late December), mainly for second years - but everyone's welcome. The notes will soon be online: `http://www.doc.ic.ac.uk/~dcw/perl2012/`
- Let's see a few examples: One commonly needed filter that doesn't exist is *sum up all these numbers*. The canonical way of doing this used to be via **awk**:

```
du -s * | awk '{t += $1} END {print t}'
```

- **Perl's** equivalent is a few characters longer and shows clearly how **awk** was a major element in the Perl language design:

```
du -s * | perl -nle '$t += $_; END {print $t}'
```

- If you use this 'sum up the numbers' one-liner very often, make it a proper Perl script:

- Create `~/bin/sumup` containing:

```
#!/usr/bin/perl -ln
$t += $_;
END {
        print $t;
}
```

- Create ~/bin/sumup containing:

```
#!/usr/bin/perl -ln
$t += $_;
END {
        print $t;
}
```

- Alternatively, expand out -n's implicit behaviour:

```
#!/usr/bin/perl -l
while( <> )
{
        $t += $_;
}
print $t;
```

- Create ~/bin/sumup containing:

```
#!/usr/bin/perl -ln
$t += $_;
END {
        print $t;
}
```

- Alternatively, expand out -n's implicit behaviour:

```
#!/usr/bin/perl -l
while( <> )
{
        $t += $_;
}
print $t;
```

- Using either of the above versions, make sumup executable by `chmod +x ~/bin/sumup`, then check the Perl syntax `perl -cw ~/bin/sumup`, then tell the shell that a new command exists (`rehash`) and finally:

```
du -s * | sumup
```

- Create `~/bin/sumup` containing:

```
#!/usr/bin/perl -ln
$t += $_;
END {
        print $t;
}
```

- Alternatively, expand out `-n`'s implicit behaviour:

```
#!/usr/bin/perl -l
while( <> )
{
        $t += $_;
}
print $t;
```

- Using either of the above versions, make `sumup` executable by `chmod +x ~/bin/sumup`, then check the Perl syntax `perl -cw ~/bin/sumup`, then tell the shell that a new command exists (`rehash`) and finally:

```
du -s * | sumup
```

- All the versions of *sum up the numbers* so far expect the numbers to be at the beginning of each line. If they're somewhere else, you need to do a bit more work to extract each number.

- Suppose the input we want to parse is the result of `find . -ls`

- Suppose the input we want to parse is the result of `find . -ls`
- Here, the file sizes are perhaps the obvious numbers to add up, and they are the seventh space-separated field in each line.

- Suppose the input we want to parse is the result of `find . -ls`
- Here, the file sizes are perhaps the obvious numbers to add up, and they are the seventh space-separated field in each line.
- To do that in a Perl one-liner, use the `-a` autosplit flag, which automatically splits each line (on whitespace) into the Perl array `@F` of which `$F[6]` is the 7th element:

```
find . -ls | perl -lane '$t += $F[6]; END {print $t}'
```

- If, instead, the data was comma-separated (i.e. in Comma-Separated-Values or CSV format) and you wanted the 3rd field, set the field separator with the `-F,` flag, as in:

```
< csv_file perl -F, -lane '$t += $F[2]; END {print $t}'
```

- Suppose the input we want to parse is the result of `find . -ls`
- Here, the file sizes are perhaps the obvious numbers to add up, and they are the seventh space-separated field in each line.
- To do that in a Perl one-liner, use the `-a` autosplit flag, which automatically splits each line (on whitespace) into the Perl array `@F` of which `$F[6]` is the 7th element:

```
find . -ls | perl -lane '$t += $F[6]; END {print $t}'
```

- If, instead, the data was comma-separated (i.e. in Comma-Separated-Values or CSV format) and you wanted the 3rd field, set the field separator with the `-F,` flag, as in:

```
< csv_file perl -F, -lane '$t += $F[2]; END {print $t}'
```

- **Perl** one-liners can also be an ad-hoc statistical toolkit: Suppose we have a list of words, one per line, and we're interested in understanding the distribution of their **lengths**:
- Let's first count the total number of characters (`$t`), number of words (`$n`) and a mapping from wordlength to frequency (`f{}`):

```
< wordlist perl -nle '$l=length($_); $t+=$l; $n++; $f{$l}++; END {print "n = $n, t = $t";
   @bylen = sort { $a <=> $b } keys %f; foreach $l (@bylen){print "$l: $f{$l}"}}'
```

- We see the minimum word length is 2, the maximum is 22, we can see the distribution right there from the numeric “graph”, in particular the modal length is clearly 8. We note that there are only two 22-letter words. What are they? Find out!

```
grep '^......................$' wordlist
```

- We see the minimum word length is 2, the maximum is 22, we can see the distribution right there from the numeric "graph", in particular the modal length is clearly 8. We note that there are only two 22-letter words. What are they? Find out!

```
grep '^......................$' wordlist
```

- Let's display a proper histogram using stars, scaling them roughly: There are 13890 words (of length 8). To scale that to about 80 characters, 13890/80 is approximately 180.

```
< wordlist perl -nle '$l=length($_); $t+=$l; $n++; $f{$l}++; END {
    print "n = $n, t = $t"; @bylen = sort { $a <=> $b } keys %f;
    foreach $l (@bylen){$stars = "*" x int($f{$l}/180); print "$l: $stars"}}'
```

- We see the minimum word length is 2, the maximum is 22, we can see the distribution right there from the numeric "graph", in particular the modal length is clearly 8. We note that there are only two 22-letter words. What are they? Find out!

```
grep '^......................$' wordlist
```

- Let's display a proper histogram using stars, scaling them roughly: There are 13890 words (of length 8). To scale that to about 80 characters, 13890/80 is approximately 180.

```
< wordlist perl -nle '$l=length($_); $t+=$l; $n++; $f{$l}++; END {
    print "n = $n, t = $t"; @bylen = sort { $a <=> $b } keys %f;
    foreach $l (@bylen){$stars = "*" x int($f{$l}/180); print "$l: $stars"}}'
```

- Finally, let's make this a nice program.. create a file (`histo.pl`) containing the following (shown as 2-columns to save space):

```
#  histo: produce word-length histogram
while( <> )
{
        chomp;
        $l = length($_);
        $t += $l;
        $n++;
        $f{$l}++;
}
```

```
@bylen = sort { $a <=> $b } keys %f;
print "n = $n, t = $t\n";
foreach $l (@bylen)
{
    $stars = "*" x int($f{$l}/180);
    print "$l: $stars\n"
}
```

and run it by `perl histo.pl < wordlist`.

- We have many useful CSG web pages, use the search box on the CSG web page:

  `http://www.doc.ic.ac.uk/csg/`

- We have many useful CSG web pages, use the search box on the CSG web page:

  `http://www.doc.ic.ac.uk/csg/`

- Google is your friend. What did we do before Google?

- We have many useful CSG web pages, use the search box on the CSG web page:

  `http://www.doc.ic.ac.uk/csg/`

- Google is your friend. What did we do before Google?
- When all else fails, you could even try the most radical option.... **RTFM** - **Read the Manual** - eg. `man gcc`, `man -k remove`.

I hope this collection of miscellaneous ways to do things the **smart way** has been useful to you!

- We have many useful CSG web pages, use the search box on the CSG web page:

  `http://www.doc.ic.ac.uk/csg/`

- Google is your friend. What did we do before Google?
- When all else fails, you could even try the most radical option.... **RTFM** - **Read the Manual** - eg. `man gcc`, `man -k remove`.

I hope this collection of miscellaneous ways to do things the **smart way** has been useful to you!

If you do only 3 things as a result of this lecture:

- Today: Sort out a sensible directory structure.

- We have many useful CSG web pages, use the search box on the CSG web page:

  `http://www.doc.ic.ac.uk/csg/`

- Google is your friend. What did we do before Google?
- When all else fails, you could even try the most radical option.... **RTFM** - **Read the Manual** - eg. `man gcc`, `man -k remove`.

I hope this collection of miscellaneous ways to do things the **smart way** has been useful to you!

If you do only 3 things as a result of this lecture:

- Today: Sort out a sensible directory structure.
- This month: Learn to use version control when it doesn't matter..

- We have many useful CSG web pages, use the search box on the CSG web page:

  `http://www.doc.ic.ac.uk/csg/`

- Google is your friend. What did we do before Google?
- When all else fails, you could even try the most radical option.... **RTFM** - **Read the Manual** - eg. `man gcc`, `man -k remove`.

I hope this collection of miscellaneous ways to do things the **smart way** has been useful to you!

If you do only 3 things as a result of this lecture:

- Today: Sort out a sensible directory structure.
- This month: Learn to use version control when it doesn't matter.. so you can use it routinely when it does!

- We have many useful CSG web pages, use the search box on the CSG web page:

  `http://www.doc.ic.ac.uk/csg/`

- Google is your friend. What did we do before Google?
- When all else fails, you could even try the most radical option.... **RTFM** - **Read the Manual** - eg. `man gcc`, `man -k remove`.

I hope this collection of miscellaneous ways to do things the **smart way** has been useful to you!

If you do only 3 things as a result of this lecture:

- Today: Sort out a sensible directory structure.
- This month: Learn to use version control when it doesn't matter.. so you can use it routinely when it does!
- Over Xmas: Learn Perl:-).